

Machine Learning for Time Series

(MLTS or MLTS-Deluxe Lectures)

Dr. Dario Zanca

Machine Learning and Data Analytics (MaD) Lab
Friedrich-Alexander-Universität Erlangen-Nürnberg
18.10.2022

Organisational Information

Machine Learning for time series

- 5 ECTS
- Lectures + Exercises

Machine Learning for Time Series (Deluxe)

- 7.5 ECTS
- Lectures + Exercises + Project

- Time series fundamentals and definitions (2 lectures) ←
- Bayesian Inference (1 lecture)
- Gaussian processes (2 lectures)
- State space models (2 lectures)
- Autoregressive models (1 lecture)
- Data mining on time series (1 lecture)
- Deep learning on time series (4 lectures)
- Domain adaptation (1 lecture)

Lectures (online)

A new lecture recording is generally released every Tuesday

Consultation hours starting on November 8th, h. 8:15 – 9:30

Exercises (online)

Live Zoom Session starting on November 9th

Recordings will be uploaded

Project (online)

Introduction during the first exercise Live Zoom Session (November 9th)

Written Exam (5 ECTS)

- Likely written or online
- 70% from lectures, 30% from exercises

Project (2.5 ECTS) - Optional

- Define your own project topics
- Work in teams of 2-3 students
- Independent work
- Write a scientific report
- Peer-review your co-student's reports

Machine Learning and Data Analytics (MaD) Lab

- Dr. Dario Zanca, dario.zanca@fau.de *
- Prof. Dr. Björn Eskofier, bjoern.eskofier@fau.de

* Please, address all your correspondence about the course to Dr. Dario Zanca

Exercises, responsibles:

- Richard Dirauf (M.Sc.), richard.dirauf@fau.de
- Philipp Schlieper (M.Sc.), philipp.schlieper@fau.de

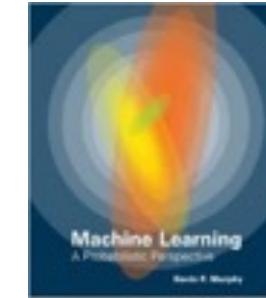
Projects, responsibles:

- Dr. Dario Zanca, dario.zanca@fau.de *
- Johannes Roider (M.Sc.), johannes.roider@fau.de

References

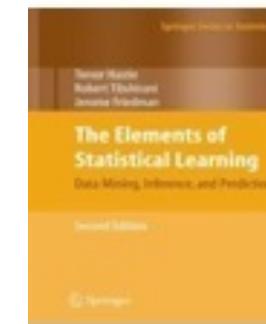
Machine learning: A Probabilistic Perspective,

by Kevin Murphy (2012)



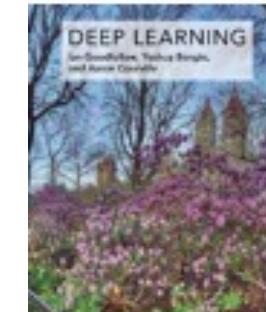
The Elements of Statistical Learning: Data Mining, Inference, and Prediction

by Trevor Hastie, Robert Tibshirani, and Jerome Friedman (2009)



Deep Learning

by Ian Goodfellow, Yoshua Bengio, and Aaron Courville (2016)





Time series fundamentals

Motivations



An old history of time series analysis: Babylonian astronomical diaries

VII century B.C.

[...] Night of the 5th, beginning of the night,
the moon was 2 $\frac{1}{2}$ cubits behind Leonis [...]

Night of the 17th, last part of the night, the
moon stood 1 $\frac{1}{2}$ cubits behind Mars, Venus
was below."

- Babylonians collected the earliest evidence of periodic planetary phenomena
- Applied their mathematics for systematic astronomic predictions



An old history of time series analysis: Babylonian astronomical diaries

Nowadays, thousands of ground-based and space-based telescopes^(a) generate new knowledge every night.

- The Vera C. Rubin Observatory in Chile is geared up to collect 20 terabytes per night from 2022^(b).
- The Square Kilometre Array, the world's largest radio telescope, will generate up to 2 petabytes daily, starting in 2028.
- The Very Large Array (ngVLA) will generate hundreds of petabytes annually.



^(a) <https://research.arizona.edu/stories/space-versus-ground-telescopes>

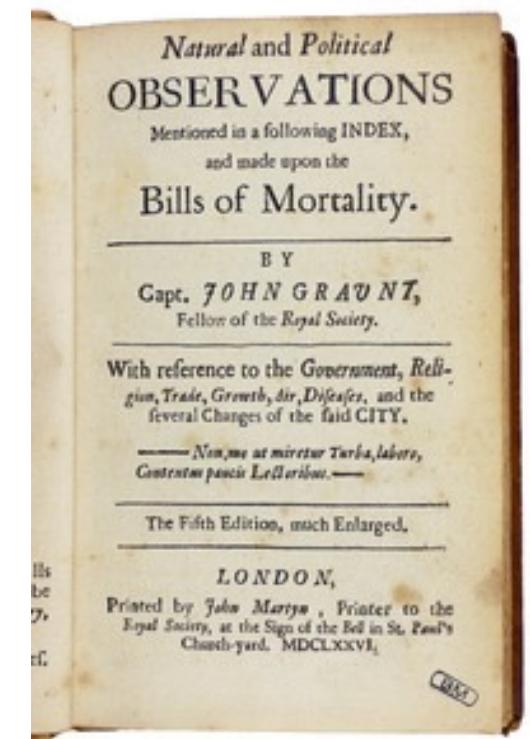
^(b) <https://www.nature.com/articles/d41586-020-02284-7>

An old history of time series analysis: The Birth of Epidemiology

1662, John Graunt describes the data collection:

"When anyone dies, [...] the same is known to the Searchers, corresponding with the said Sexton. The Searchers hereupon...examine by what Disease, or Casualty the corps died. Hereupon they make their Report to the Parish-Clerk, and he, every Tuesday night, carries in an Accompt of all the Burials, and Christnings, hapning that Week, to the Clerk of the Hall."

- Rudimentary conclusions about the mortality and morbidity of certain diseases
- Graunt's work is still used today to study population trends and mortality

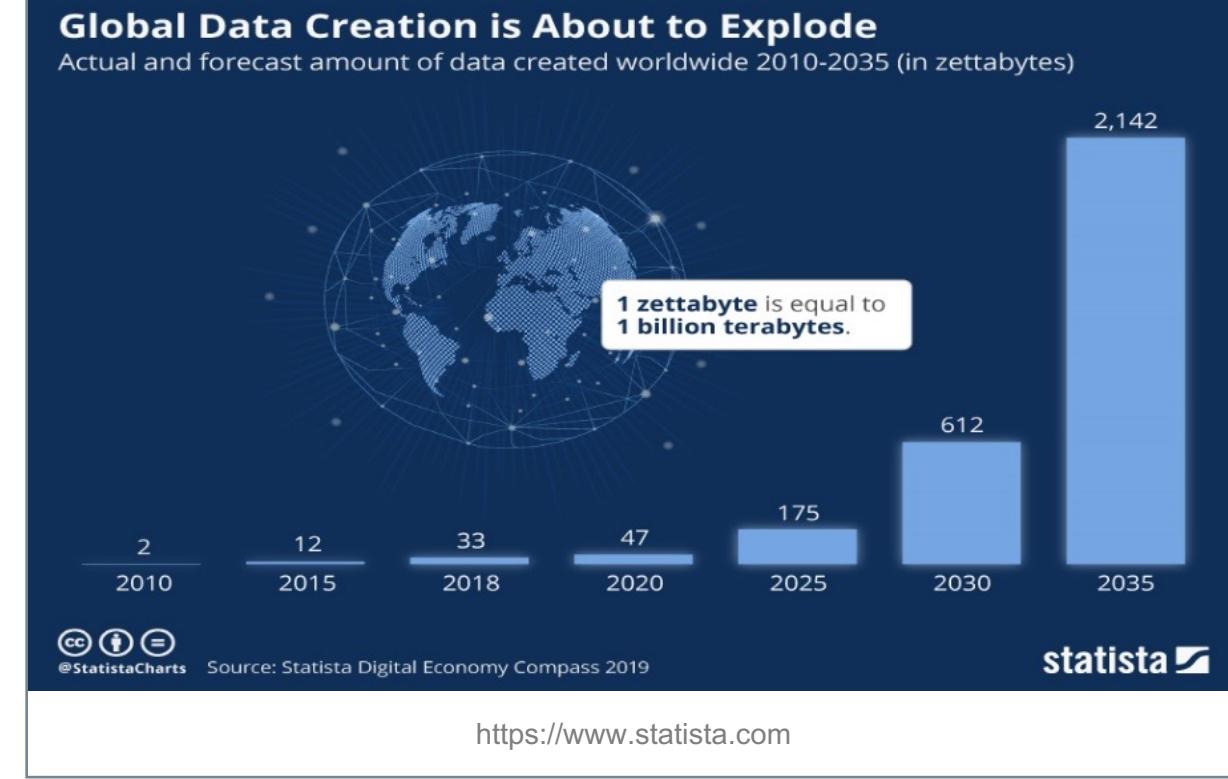


Importance of time series

Machine learning on time series is becoming increasingly important because of the massive production of time series data from diverse sources, e.g.,

- Digitalization in healthcare
- Internet of things
- Smart cities
- Process monitoring

The amount of created data increased from two zettabytes in 2010 to 47 zettabytes in 2020

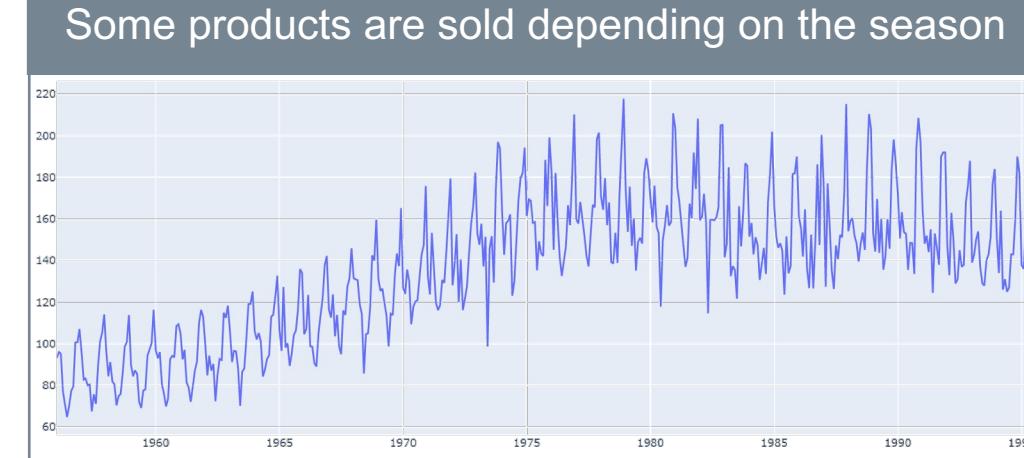
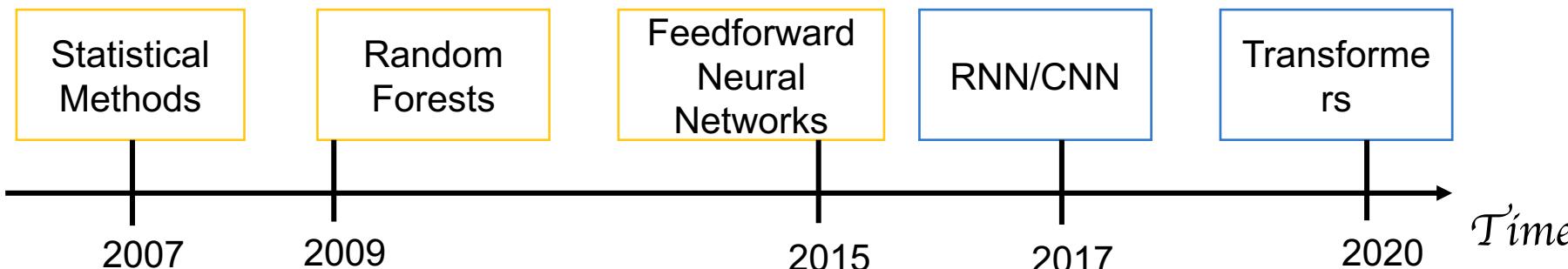


Example: Predicting demand of **amazon** products

Amazon sells 400 million products in over 185 countries^(a).

- Maintaining surplus inventory levels for every product is cost-prohibitive.
- Predict future demand of products

Methods:

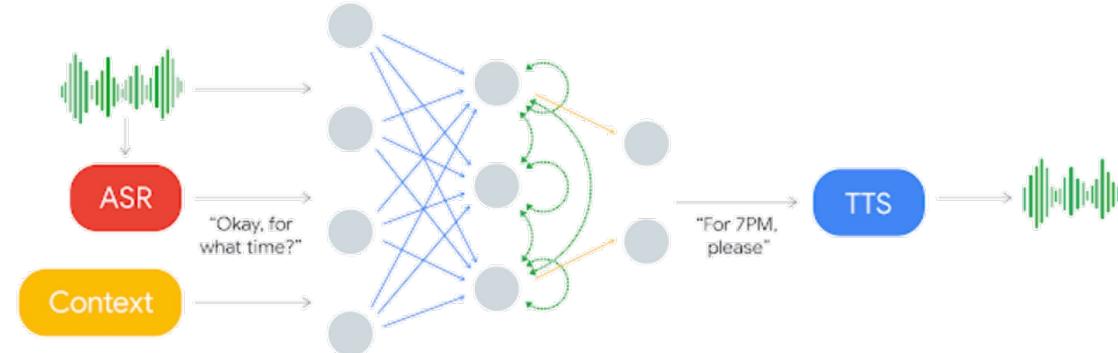


- First models required manual feature engineering
- New methods are fully data-driven

Example: Google Duplex makes tedious phone calls

Method: An RNNs with several features. We use a combination of text to speech (TTS) engine and a synthesis TTS engine to control intonation (e.g., “hmm”s and “uh”s).

Limitations: trained on specific tasks. Cannot deal general conversations.



- Additional audio features
- Automatic speech recognition
- Desired service, time/day

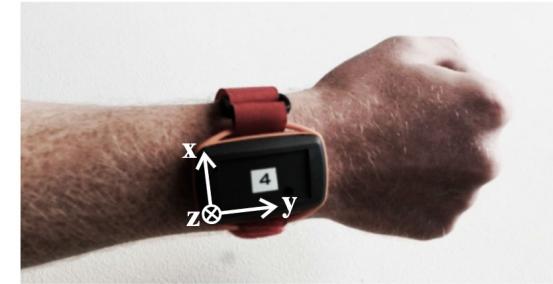


E.g., Duplex calling a restaurant.

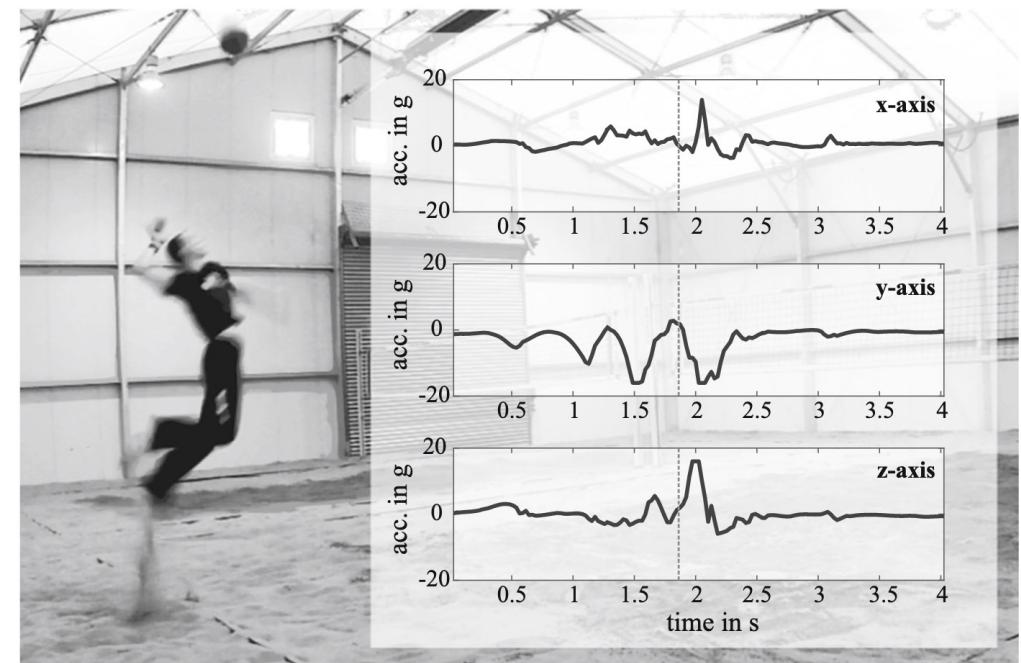
Example: Activity recognition in sports (FAU Erlangen)

Many injuries in sports are caused by overuse.

- These injuries are a major cause for reduced performance of professional and non-professional beach volleyball players.
- Monitoring of player actions could help identifying and understanding risk factors and prevent such injuries.



Sensor attachment at the wrist of the dominant hand with a soft, thin wristband

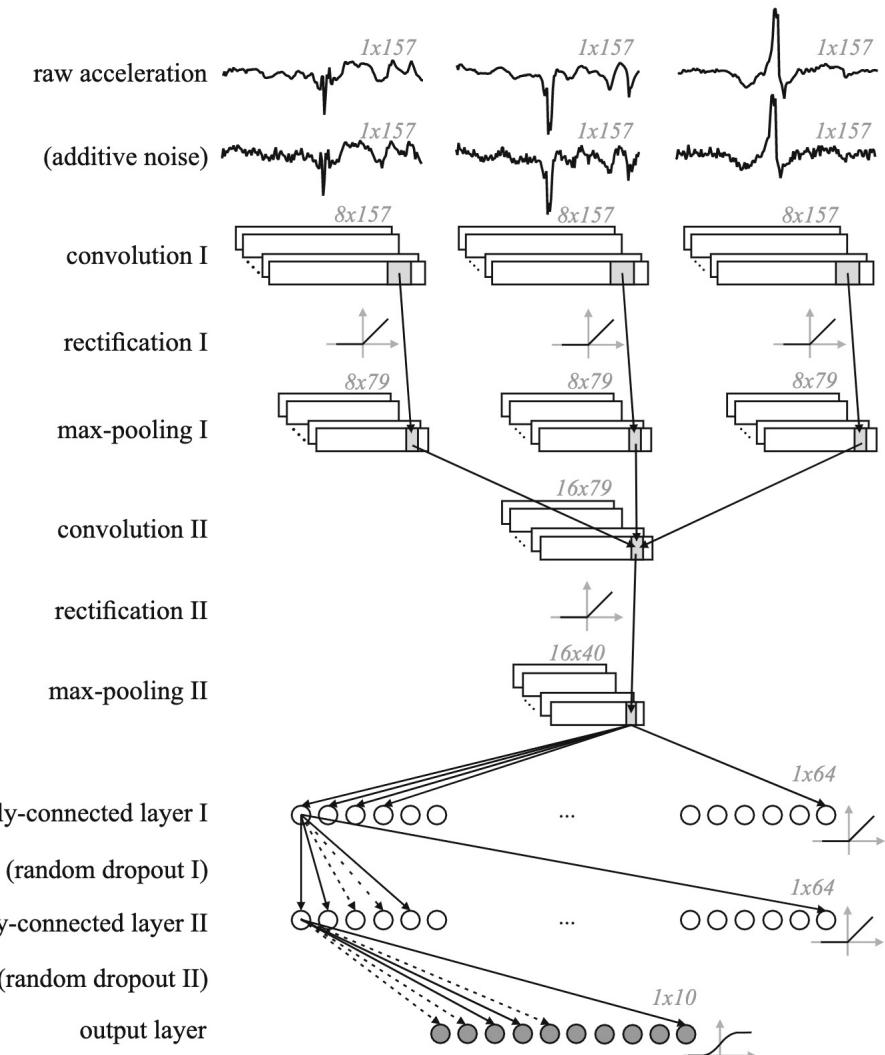


Example: Activity recognition in sports (FAU Erlangen)

Method: A CNN is used to classify players' activities. Classifications allow to create players' profiles.

Actions:

- Underhand serve
- Overhand serve
- Jump serve
- Underarm set
- Overhead set
- Shot attack
- Spike
- Block
- Dig
- Null class.





Time series fundamentals

Definitions and basic properties



What is a time series?

A time series can be described as a set of observations, taken sequentially in time,

$$S = \{s_1, \dots, s_T\}$$

where $s_i \in \mathbb{R}^d$ is the measured state of the observed process at time t_i .

Typically, observations are generally *dependent*

- Studying the nature of this dependency is of particular interest
- Time series analysis is concerned with techniques for the analysis of these dependencies

Terminology: Regularly Sampled vs Irregularly Sampled

Discrete time series are **regularly sampled** if their observations are equally spaced in time.

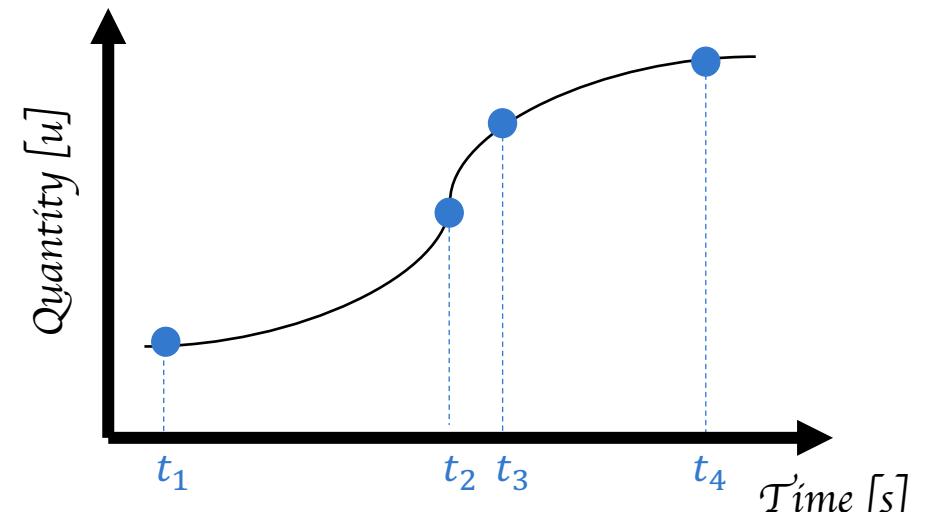
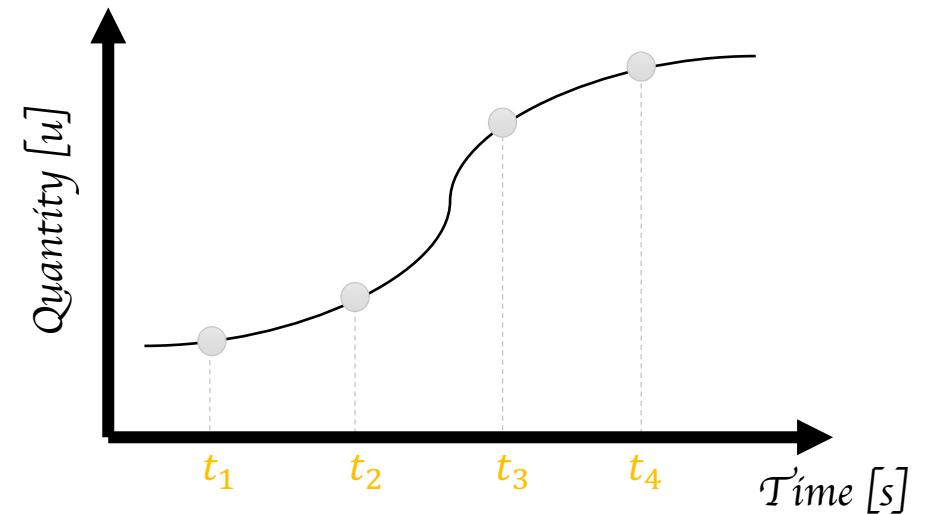
$$\forall i \in \{1, \dots, T - 1\},$$

$$\Delta t_i = t_{i+1} - t_i = \text{const.}$$

In contrast, for **irregularly sampled** time sequences, the observations are not equally spaced.

- They are generally defined as a collection of pairs

$$S = \{(s_1, t_1), \dots, (s_T, t_T)\}$$



Terminology: Univariate vs Multivariate

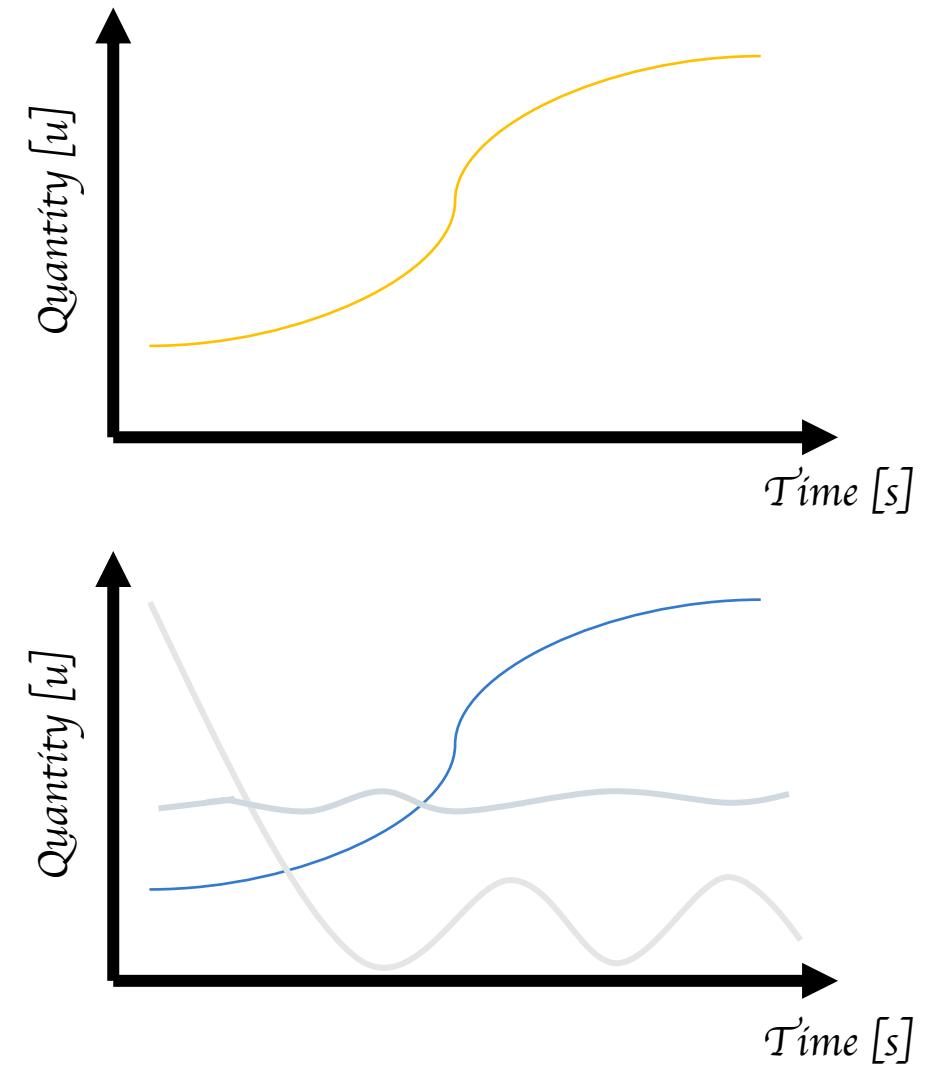
Let $S = (s_1, \dots, s_T)$ be a time series,
where $s_i \in \mathbb{R}^d, \forall i \in \{1, \dots, T\}$.

If $d = 1$, S is said **univariate**.

- Only one variable is varying over time.

If $d > 1$, S is said **multivariate**.

- Multiple variables are varying over time
 - E.g., tri-axial accelerometer measurements



Terminology: Discrete vs Continuous

A time series is said to be **continuous** if observations are made at each instant of time, even when its measurements consist only of a discrete set of values.

- E.g., the number of people in a room.

A time series is said to be **discrete** if observations are taken at specific times. Discrete time series can arise in different ways:

- Sampled (e.g., daily rainfall)
- Aggregated (e.g., monthly reports of daily rainfalls)

Terminology: Discrete vs Continuous

We will denote as **mixed-type** a multivariate time series consisting of both continuous and discrete observations

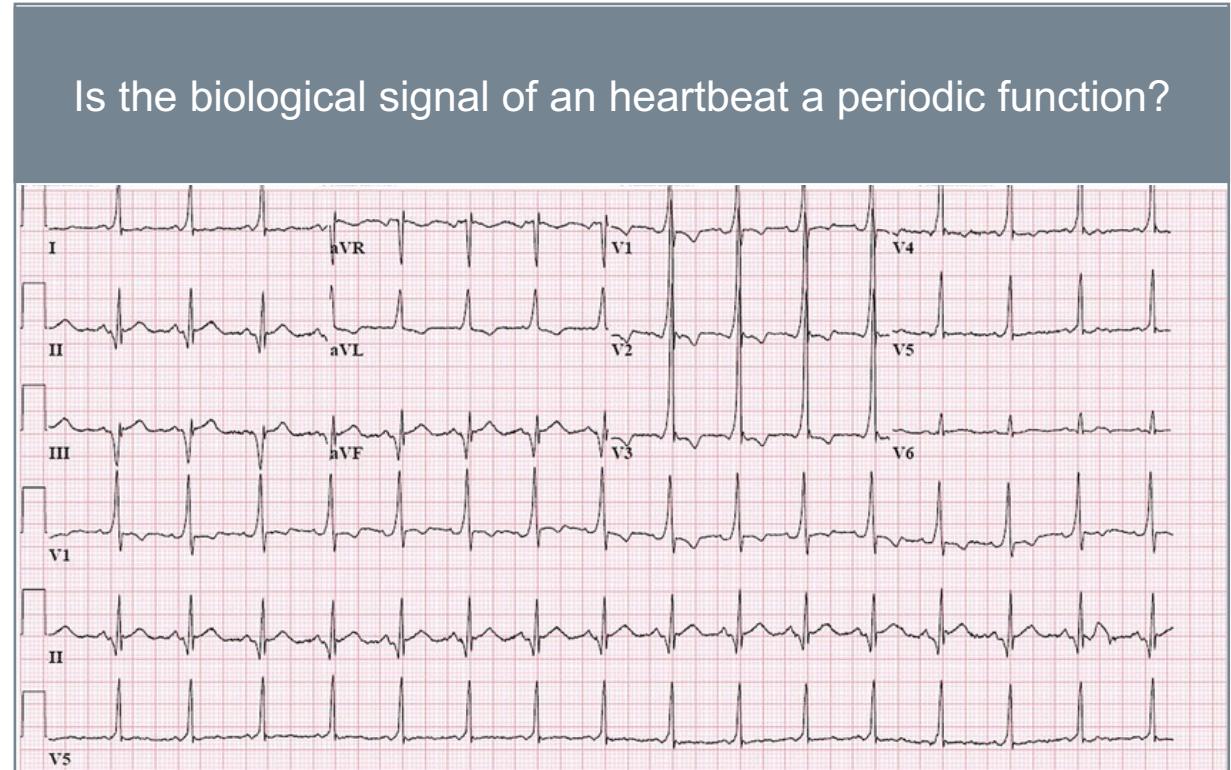
- E.g., a time series consisting of continuous sensor values and discrete event log for the monitoring of an industrial machine

Terminology: Periodic

A time series is said **periodic** if there exists a number $\tau \in \mathbb{R}$, called *period*, such that

$$s_i = s_{i+\tau}, \forall i \in \{1, \dots, T - \tau\}$$

E.g., the continuous time series defined by the trigonometric function $f(x) = \sin(x)$



Terminology: Deterministic vs Non-Deterministic

A **deterministic** time series is one that could be expressed explicitly by an analytical expression.

- Observations are generated from a system with no randomness.

In contrast, a **non-deterministic** time series can not be described by an analytic expression. A time series may be non-deterministic because :

- The information necessary to describe the process is not fully observable, or
- The process generating the time series is inherently random

Stochastic Process

Non-deterministic time series can be regarded as manifestations (equiv., realization) of a **stochastic process**, which is defined as a set of random variables $\{X_t\}_{t \in \{1, \dots, T\}}$

Even if we were to imagine having observed the process for an infinite period T of time, the infinite sequence

$$S = \{\dots, s_{t-1}, s_t, s_{t+1}, \dots\} = \{s_t\}_{t=-\infty}^{+\infty}$$

would still be a single **realization** from that process.

Stochastic Process

Still, if we had a battery of N computers generating series $S^{(1)}, \dots, S^{(N)}$, and considering selecting the observation at time t from each series,

$$\{s_t^{(1)}, \dots, s_t^{(N)}\}$$

this would be described as a sample of N realizations of the random variable X_t

This random variable X_t is associated with an **unconditional density**, denoted by

$$f_{X_t}(s_t)$$

- E.g., for the Gaussian white noise process $f_{X_t}(s_t) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-s_t^2}{2\sigma^2}}$

Stochastic Process

The **unconditional mean** is the expectation, provided it exists, of the t -th observation, i.e.,

$$E(X_t) = \int_{-\infty}^{+\infty} s_t f_{X_t}(s_t) ds_t = \mu_t$$

Similarly, the **variance** of the random variable X_t is defined as

$$E(X_t - \mu_t)^2 = \int_{-\infty}^{+\infty} (s_t - \mu_t)^2 f_{X_t}(s_t) ds_t$$

Stochastic Process

Given any particular realization $S^{(i)}$ of a stochastic process (i.e., a time series), we can define the vector of the $j + 1$ most recent observations

$$x_t^i = [s_{t-j}^{(i)}, \dots, s_t^{(i)}]$$

We want to know the probability distribution of this vector x_t^i across realizations. We can calculate the **j -th autocovariance**

$$\gamma_{jt} = E(X_t - \mu_t)(X_{t-j} - \mu_{t-j})$$

Stationarity

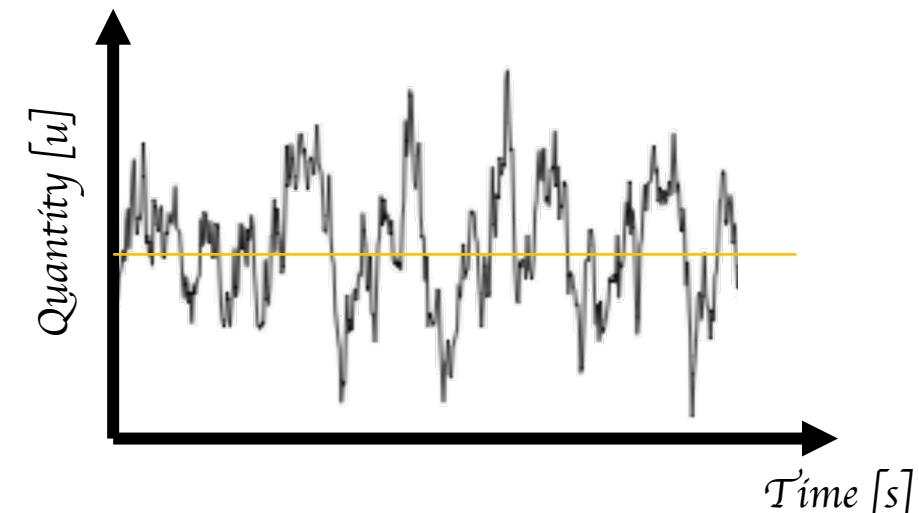
If neither the mean μ_t or the autocovariance γ_{jt} depend on the temporal variable t , then the process is said to be (weakly) **stationary**.

E.g., let the stochastic process $\{X_t\}_{t=-\infty}^{+\infty}$ represent the sum of a constant μ with a Gaussian white noise process $\{\epsilon_t\}_{t=-\infty}^{+\infty}$, such that

$$X_t = \mu + \epsilon_t$$

Then, its mean is constant: $E(X_t) = \mu + E(\epsilon_t) = \mu$

and its j -th autocovariance: $E(X_t - \mu)(X_{t-j} - \mu) = \gamma_j$



In other words: A process is said to be stationary if the process statistics do not depend on time.

Ergodicity

Given a time series, denoted by $S^{(i)} = \{s_1^{(i)}, \dots, s_T^{(i)}\}$, we can compute the sample temporal average as

$$\bar{s} = \frac{1}{T} \sum_{t=1}^T s_t^{(i)}$$

The ergodicity of a time series bind the concept of the process mean with that of temporal sample mean:

- A process is said to be ergodic if \bar{s} converges to μ_t as $T \rightarrow \infty$

In other words: A process is said to be ergodic if its time statistics equals the process statistic, provided that the process is observed long enough.

Example: Stationarity and Ergodicity

To clarify the concept, we give an example of stationary but not ergodic process. Suppose the mean $\mu^{(i)}$ of the i -th realization of $\{X_t\}_{t=-\infty}^{+\infty}$ is sampled from the normal distribution $U(0, \lambda^2)$ and, similarly to the previous example, $X_t^{(i)} = \mu^{(i)} + \epsilon_t$.

We have that the process is stationary because:

$$\mu_t = E(\mu^{(i)}) + E(\epsilon_t) = 0$$

$$\gamma_{jt} = E(\mu^{(i)} + \epsilon_t)(\mu^{(i)} + \epsilon_{t-j}) = \lambda^2$$

Example: Stationarity and Ergodicity

However, its sample temporal mean, converges to a different value than the process mean, i.e.,

$$\bar{s} = (1/T) \sum (\mu^{(i)} + \epsilon_t) = \mu^{(i)}$$



Time series fundamentals

i.i.d. observations and central limit theorems



Time series and i.i.d. data

Observations collected in a time series $S = (s_1, \dots, s_T)$ are **generally not i.i.d.**

- Observation s_i could be **dependent** on previous observations s_j , with $j < i$
- The distribution of the underlying data generation process could change over time, i.e. it is **not identically distributed**

For example:

- The price of a stock today depends on its price yesterday (**dependence**)
- and the volatility of the stock, i.e., its dispersion of returns, might change over time (**change on the underlying distribution**)

Time series and i.i.d. data

The structure of this dependence imposes challenges on the statistical data analysis of time series.

- Many tools for statistical inference are valid only for i.i.d. data

Time series and i.i.d. data

It might be useful to be able to assess the structure of the dependence between random variables. For this reason we make use of their correlation.

- Generally, we measure the correlation between two variables X_i and X_j with their **covariance** $\text{Cov}(X_i, X_j)$.
 - $\text{Cov}(X_i, X_j) = 0 \rightarrow$ uncorrelated
- We measure dependence of an entire time series with a similar concept, the **long-run variance**
 - $\sigma_i^2 = \sum_{\mathbb{Z}} \text{Cov}(X_i, X_{i+h})$

The Central Limit Theorem

The **Central Limit Theorem (CLT)** suggests that the sum of random variables converges to a normal distribution, under precise conditions.

More precisely, for a sequence of i.i.d. random variables $\{X_t\}_{t \in \{1, \dots, T\}}$ with $\mu = E(X_t)$ and $\sigma^2 = E(X_t - \mu)^2$, by the CLT it holds:

$$\sqrt{T} \left(\frac{1}{T} \sum_1^T X_i - \mu \right) \rightarrow \mathcal{N}(0, \sigma^2)$$

For stationary time series with mean μ and long-run variance σ^2 the CLT holds as before.

Why is the CLT important?

If the CLT holds for a time series, we can draw from a larger range of methods.

- Statistical inference depends on the possibility to take a broad view of results from a sample to the population.
- The CLT legitimizes the assumption of normality of the error terms in linear regression.

However,

- Many time series we encounter in the real world satisfy CLT assumption of independence and stationarity
- Or can be transformed into stationary time series, e.g., by differentiations or other transformations

It is a good idea to start by checking whether the data is independent or stationary.

Insight: CLT for dependent random variables

Different version of the CLT exist for dependent random variables. For example, under the assumption of a M-dependent random process^(a), we have that the following limit theorem holds:

Let $\{X_t\}_{t \in \{1, \dots, T\}}$ be M-dependent stationary process with mean μ , covariance γ_j , and denoted with V_M the variance of the mean of n observations,

$$V_M := \sum_{j=-M}^M \gamma_j$$

If $V_M > 0$, then,

$$\sqrt{n}(X_i - \mu) \rightarrow N(0, V_M).$$

^(a) A stochastic process $\{X_t\}_{t \in \{1, \dots, T\}}$ is said to be M-dependent if $\{X_t\}_{t \leq k}$ are independent of the stochastic variables $\{X_t\}_{t \geq k+M+1}$



Time series fundamentals

Recap



Recap

Time series have long been studied in history

- Recent digitalization increases the importance of time series analysis

Properties of time series

- Regularly vs irregularly sampled
- Univariate vs multivariate
- Discrete vs continuous
- Periodic
- Deterministic vs non-deterministic
- Stationarity
- Ergodicity

Central limit theorem only holds for stationary time series

- Less restrictive CLT versions exist
- Need to properly learn dependences





Machine Learning for Time Series

(MLTS or MLTS-Deluxe Lectures)

Dr. Dario Zanca

Machine Learning and Data Analytics (MaD) Lab
Friedrich-Alexander-Universität Erlangen-Nürnberg
25.10.2022

Organisational Information

Lectures (online)

A new lecture recording every Tuesday.

Consultation hours from November 8th, h. 8:15 – 9:30

Exercises (online)

Live Zoom Session starting on November 9th

Recordings will be uploaded

Project (online)

Introduction during the first exercise Live Zoom Session (November 9th)

Applications starting on Nov 9th

- Time series fundamentals and definitions (2 lectures) ←
- Bayesian Inference (1 lecture)
- Gaussian processes (2 lectures)
- State space models (2 lectures)
- Autoregressive models (1 lecture)
- Data mining on time series (1 lecture)
- Deep learning on time series (4 lectures)
- Domain adaptation (1 lecture)

In this lecture...

1. Types of Machine Learning
2. ML pipeline and good practise
3. ML tasks with time series
4. Example: linear regression

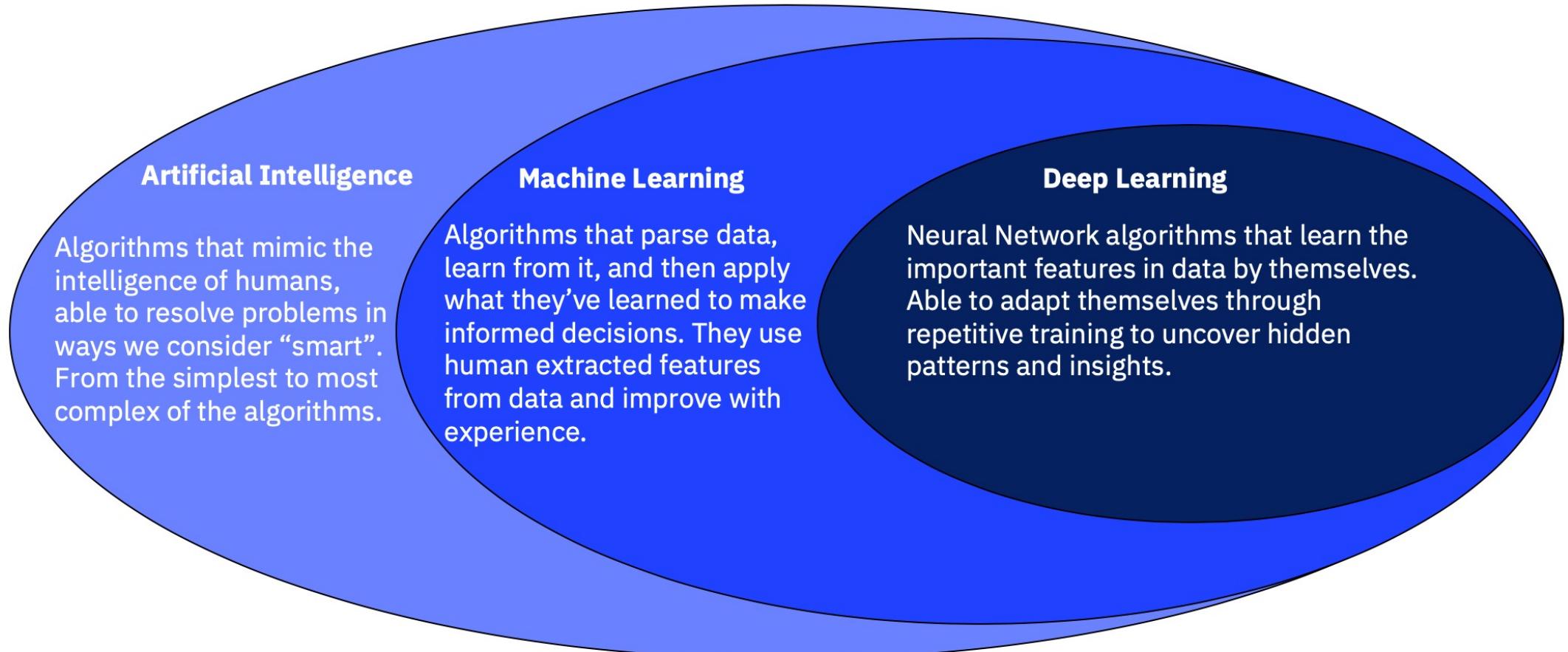


Machine learning basics

Types of machine learning



What is Machine Learning (ML)?



Supervised, unsupervised and reinforcement learning

Supervised Learning

- Learning using a teacher
- Makes machine learning explicitly
- Labeled data

Unsupervised Learning

- Learning using an “abstract” metric
- Machine understands the data (Identifies patterns/structures)
- Evaluation is qualitative or indirect

Reinforcement Learning

- Reward based learning
- Learning from positive and negative reinforcement
- Machine learns how to act in a certain environment to maximize rewards

Supervised, unsupervised and reinforcement learning

Supervised Learning

- Learning using a teacher
- Makes machine learning explicitly
- Labeled data

Unsupervised Learning

- Learning using an “abstract” metric
- Machine understands the data (Identifies patterns/structures)
- Evaluation is qualitative or indirect

Reinforcement Learning

- Reward based learning
- Learning from positive and negative reinforcement
- Machine learns how to act in a certain environment to maximize rewards

Supervised Learning

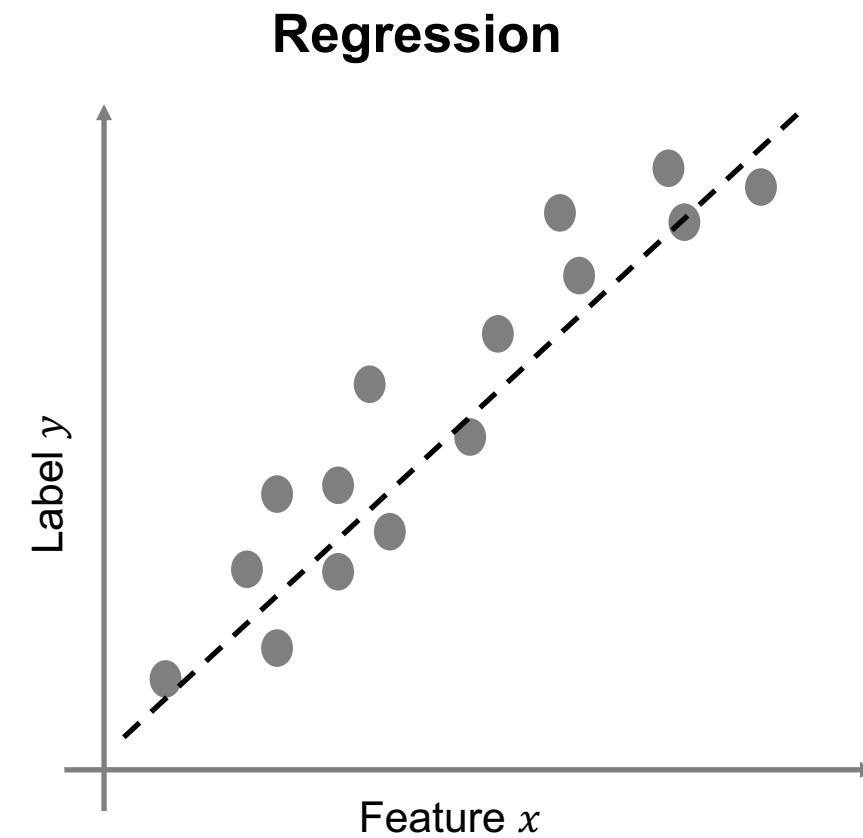
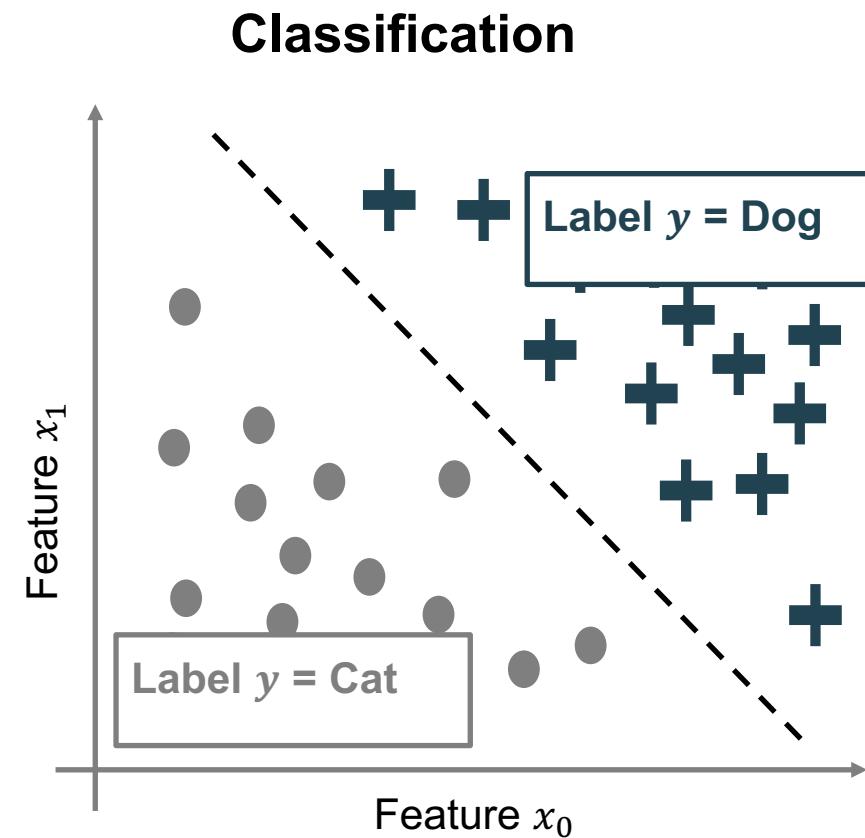
The agent observes some example **Input (Features)** and **Output (Label)** pairs and learns a **function that maps input to output**.

Key Aspects:

- Learning is **explicit**
- Learning using **direct feedback**
- Data with **labeled output**

→ Resolves classification and regression problems

Supervised Learning Problems



Regression

Regression is used to predict
a **continuous value**

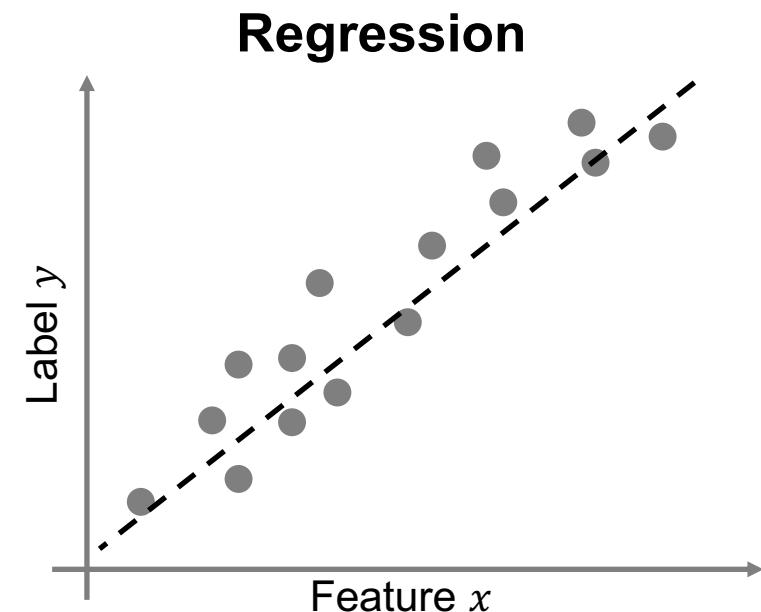
Training is based on a set of
input – output pairs (**samples**)

$$\mathcal{D} = \{(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)\}$$

Sample : (x_i, y_i)

$x_i \in \mathbb{R}^m$ is the **feature vector** of sample i

$y_i \in \mathbb{R}$ is the **label value** of sample i



Regression

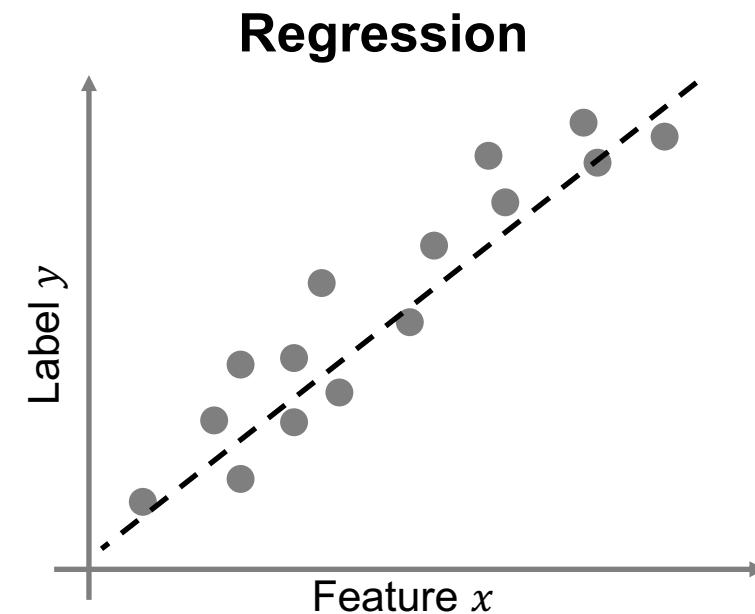
Goal:

Find a relationship (function), which expresses the input and output the best!

That means, we **fit a regression model** f to all samples:

$$f(x_i) = y_i \quad , \forall (x_i, y_i) \in \mathcal{D}$$

In this case f is a **linear regression model!**
(Black Line)



Classification

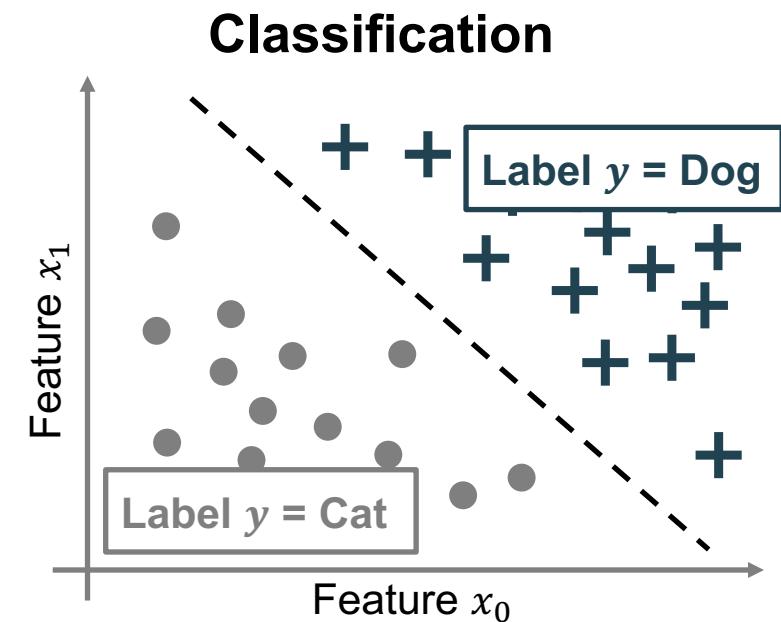
Classification is used to predict
the **class** of the input

Sample : $s_i = (\vec{x}_i, \vec{y}_i)$

$\vec{y}_i \in L$ is the **label** of sample i

In this example:

- $L = \{Cat, Dog\}$
- Binary classification problem
- The output belongs to only one class!



Classification

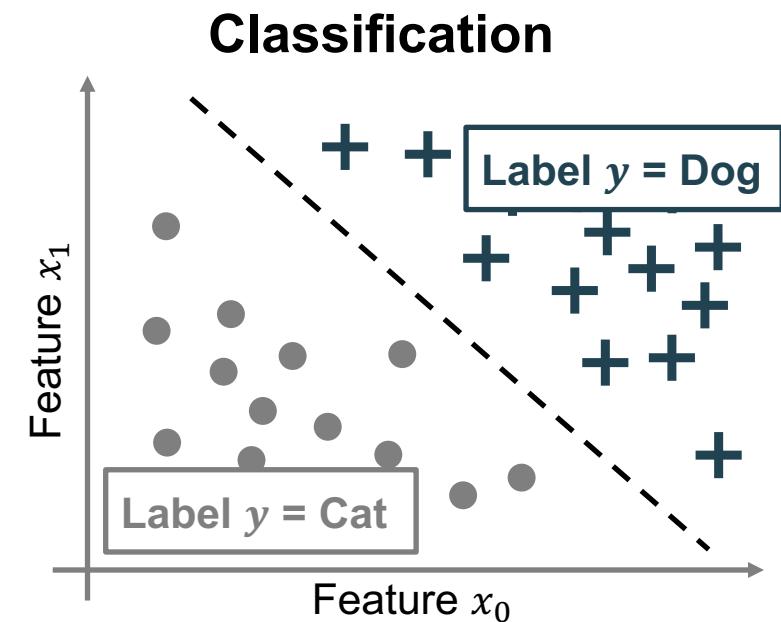
Goal:

Find a way to divide the input into the output classes!

That means, we **find a decision boundary f** for all samples:

$$f(x_i) = y_i \quad , \forall (x_i, y_i) \in \mathcal{D}$$

In this case f is a **linear decision boundary!**
(Black Line)



Supervised Learning Problems

Regression	Classification
<ul style="list-style-type: none">The output are continuous or real values	<ul style="list-style-type: none">The output variable must be a discrete value (class)
<ul style="list-style-type: none">We try to fit a regression model, which can predict the output more accurately	<ul style="list-style-type: none">We try to find a decision boundary, which can divide the dataset into different classes.
<ul style="list-style-type: none">Regression algorithms can be used to solve the regression problems such as Weather Prediction, House price prediction, Stock market prediction etc.	<ul style="list-style-type: none">Classification Algorithms can be used to solve classification problems such as Hand-written digits(MNIST), Identification of cancer cells, Defected or Undefected solar cells etc.

Supervised, unsupervised and reinforcement learning

Supervised Learning

- Learning using a teacher
- Makes machine learning explicitly
- Labeled data

Unsupervised Learning

- Learning using an “abstract” metric
- Machine understands the data (Identifies patterns/structures)
- Evaluation is qualitative or indirect

Reinforcement Learning

- Reward based learning
- Learning from positive and negative reinforcement
- Machine learns how to act in a certain environment to maximize rewards

Unsupervised Learning

Unsupervised learning observes some example Input (Features) – No Labels! - and finds patterns based on a metric

Key Aspects:

- Learning is **implicit**
- Learning using **indirect feedback**
- Methods are **self-organizing**

Resolves **clustering** and **dimensionality reduction** problems

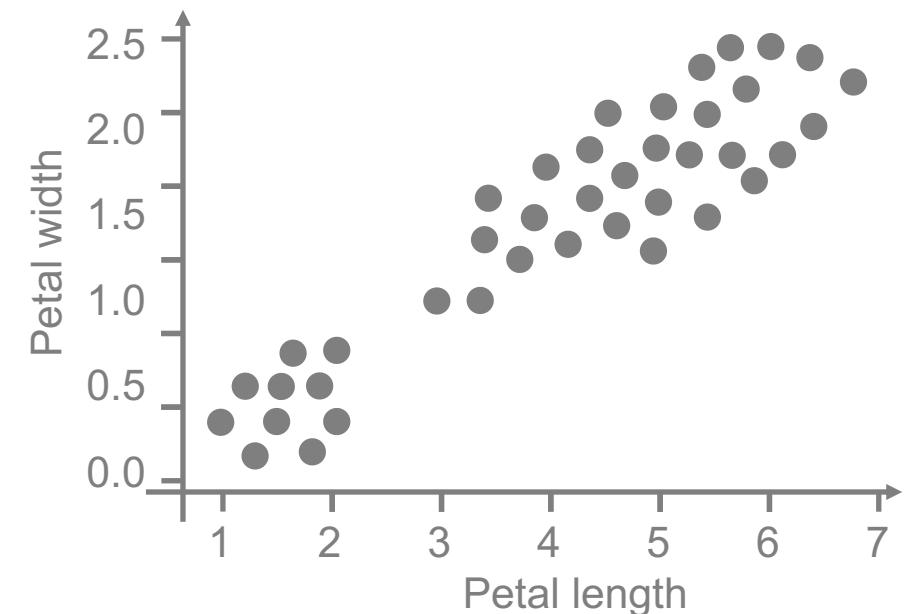
Clustering

Goal: Identify similar samples and assign them the same label

Mostly used for data analysis,
data exploration, and/or
data preprocessing

Clustering basic principles:

- Homogeneous data in the cluster
(Intra-cluster distance)
- Heterogenous data between the cluster
(Inter-cluster distance)



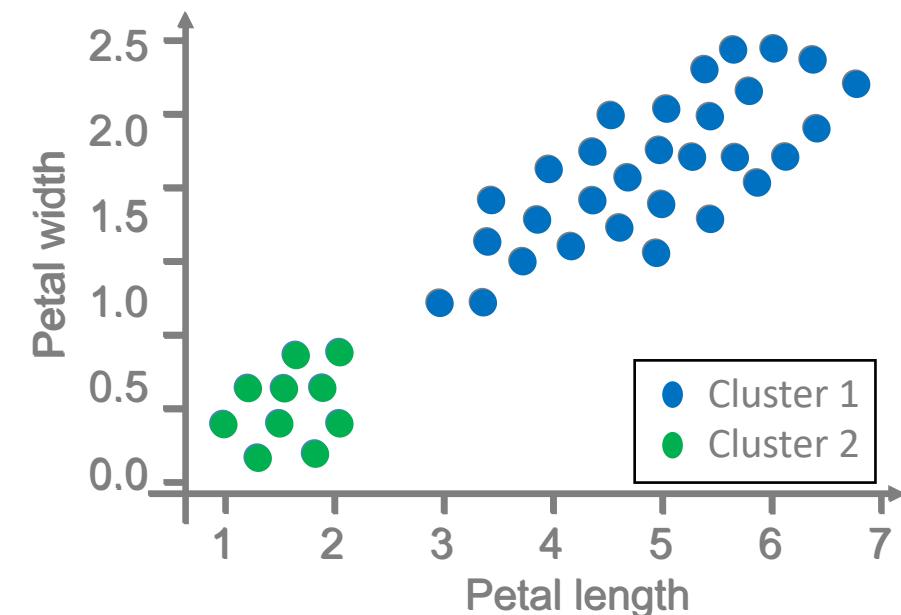
Clustering

Goal: Identify similar samples and assign them the same label

Mostly used for data analysis,
data exploration, and/or
data preprocessing

Clustering basic principles:

- Homogeneous data in the cluster
(Intra-cluster distance)
- Heterogenous data between the cluster
(Inter-cluster distance)



Curse of dimensionality

As the number of features or dimensions grows, the amount of data we need to generalize accurately grows exponentially.” – Charles Isbell

The intuition in lower dimensions does not hold in higher dimensions:

- Almost all samples are close to at least one boundary
- Distances (e.g., Euclidean) between all samples are similar
- Features might be wrongly correlated with outputs
- Finding decision boundaries becomes more complex

→ Problems become much more difficult to solve!

Example: Curse of dimensionality

The production system has N sensors attached with either the input set to “On” or “Off”

Question: How many samples do we need, to have **all possible sensor states** in the dataset?

$$N = 1 : |D| = 2^1 = 2$$

$$N = 10 : |D| = 2^{10} = 1024$$

$$N = 100 : |D| = 2^{100} = 1.2 \times 10^{30}$$

For $N = 100$, the number of points are even more than the number of atoms in the universe!

Dimensionality reduction

The goal:

Transform the samples from a high to a lower dimensional representation!

Ideally:

Find a representation, which solves your problem!

Typical Approaches:

- Feature Selection
- Feature Extraction

	S0	S1	S2	S3	S4	S5	S6	S7	S8
Sample0	0.2	0.1	11.1	2.2	Off	7	1.1	0	1.e-1
Sample1	1.2	-0.1	3.1	-0.1	On	9	2.3	-1	1.e-4
Sample2	2.7	1.1	0.1	0.1	Off	10	4.5	-1	1.e-9
Sample3	3.1	0.1	1.1	0.2	Off	1	6.6	-1	1.e-1

	T0	T1	T2	T3
Sample0	11.3	0.1	-1	7.8
Sample1	4.3	-0.1	1	6.8
Sample2	2.8	1.1	-1	7.1
Sample3	4.2	0.1	1	6.9

Dimensionality reduction

The goal:

Transform the samples from a high to a lower dimensional representation!

Ideally:

Find a representation, which solves your problem!

Typical Approaches:

- Feature Selection
- Feature Extraction



	S0	S1	S2	S3	S4	S5	S6	S7	S8
Sample0	0.2	0.1	11.1	2.2	Off	7	1.1	0	1.e-1
Sample1	1.2	-0.1	3.1	-0.1	On	9	2.3	-1	1.e-4
Sample2	2.7	1.1	0.1	0.1	Off	10	4.5	-1	1.e-9
Sample3	3.1	0.1	1.1	0.2	Off	1	6.6	-1	1.e-1
⋮									

Identical

	T0	T1	T2	T3
Sample0	11.3	0.1	-1	7.8
Sample1	4.3	-0.1	1	6.8
Sample2	2.8	1.1	-1	7.1
Sample3	4.2	0.1	1	6.9
⋮				

	T0	T1	T2	T3
Sample0	11.3	0.1	-1	7.8
Sample1	4.3	-0.1	1	6.8
Sample2	2.8	1.1	-1	7.1
Sample3	4.2	0.1	1	6.9
⋮				

Dimensionality reduction

The goal:

Transform the samples from a high to a lower dimensional representation!

Ideally:

Find a representation, which solves your problem!

Typical Approaches:

- Feature Selection
- Feature Extraction



	S0	S1	S2	S3	S4	S5	S6	S7	S8
Sample0	0.2	0.1	11.1	2.2	Off	7	1.1	0	1.e-1
Sample1	1.2	-0.1	3.1	-0.1	On	9	2.3	-1	1.e-4
Sample2	2.7	1.1	0.1	0.1	Off	10	4.5	-1	1.e-9
Sample3	3.1	0.1	1.1	0.2	Off	1	6.6	-1	1.e-1
⋮									

Applied a function f

	T0	T1	T2	T3
Sample0	11.3	0.1	-1	7.8
Sample1	4.3	-0.1	1	6.8
Sample2	2.8	1.1	-1	7.1
Sample3	4.2	0.1	1	6.9
⋮				

Supervised, unsupervised and reinforcement learning

Supervised Learning

- Learning using a teacher
- Makes machine learning explicitly
- Labeled data

Unsupervised Learning

- Learning using an “abstract” metric
- Machine understands the data (Identifies patterns/structures)
- Evaluation is qualitative or indirect

Reinforcement Learning

- Reward based learning
- Learning from positive and negative reinforcement
- Machine learns how to act in a certain environment to maximize rewards

Reinforcement Learning

Reinforcement learning observes some example Input (Features) – No Labels! - and finds the **optimal action** i.e., maximizes its future reward

Key Aspects:

- Learning is **implicit**
- Learning using **indirect feedback** based on trials and reward signals
- Actions are **affecting future measurements (i.e., inputs)**

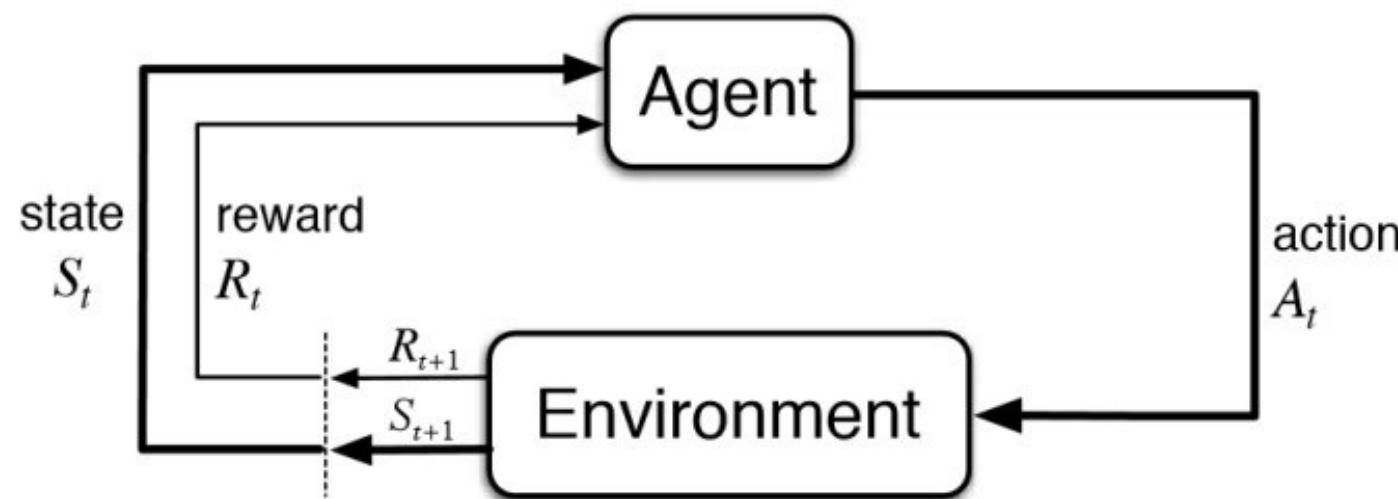
Resolves **control** and **decision** problems

- i.e., controlling agents in games or robots

Reinforcement Learning

Goal: Agents should take actions in an environment which maximize the cumulative reward.

To achieve this RL uses **reward and punishment** signals based on the previous actions to optimize the model.



Reinforcement Learning vs Unsupervised Learning

Unsupervised Learning	Reinforcement Learning
<ul style="list-style-type: none">An indirect feedback is generated according to a metric	<ul style="list-style-type: none">The feedback is given by a reward signal
<ul style="list-style-type: none">Feedback is instantaneous	<ul style="list-style-type: none">Feedback can be delayed (credit assignment problem)
<ul style="list-style-type: none">Learning by using static data (no re-recording of data necessary)	<ul style="list-style-type: none">Training is based on trials i.e. interaction between environment and agent (re-recording necessary)
<ul style="list-style-type: none">Prediction does not affect future measurements – The data is assumed Independent Identically Distributed (i.i.d)	<ul style="list-style-type: none">The prediction (actions) affect future measurements i.e. the measurements are no necessarily i.i.d!



Machine learning basics

ML pipeline and good practices



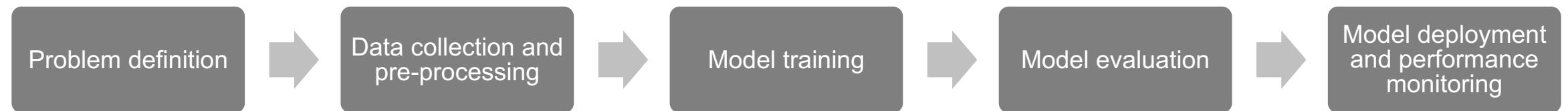
The ML pipeline

The concept of a pipeline guidance in a machine learning project.

- step-by-step process
- Each step has a **goal** and a **method**

There exist many pipelines proposals in the literature.

Here we propose a compact 5-steps pipeline:



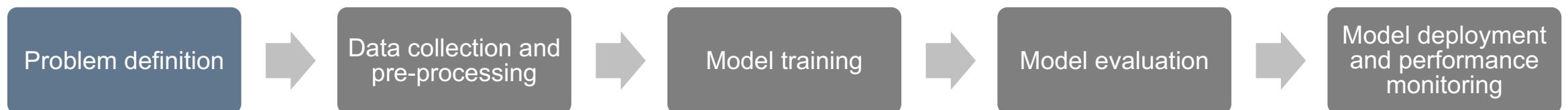
Step 1. Problem definition

In order to develop a satisfying solution, we need to define the problem.

- What goal (or **task**) we want to solve
- What kind of **data** we need

E.g., our goal is to monitor an industrial machine to predict failure and allow convenient scheduling of corrective maintenance.

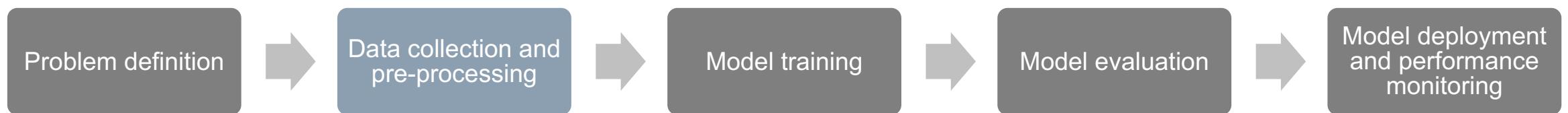
- Our goal is to predict failure one week in advance
 - Alternatively, predict the remaining useful life
- Prediction is based on data from the machine (sensors) and users (logs)



Step 2. Data collection and pre-processing

The phase of gathering the data and creating our dataset is called **data ingestion**.

- Data should **contain necessary information** to solve the task
- Data should **be enough** to describe all possible states



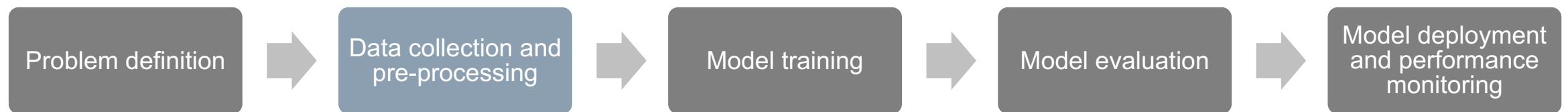
Step 2. Data collection and pre-processing

The phase of gathering the data and creating our dataset is called **data ingestion**.

- Data should **contain necessary information** to solve the task
- Data should **be enough** to describe all possible states

Then, a **data preparation** phase follows, with the goal of making data usable for our machine learning solution.

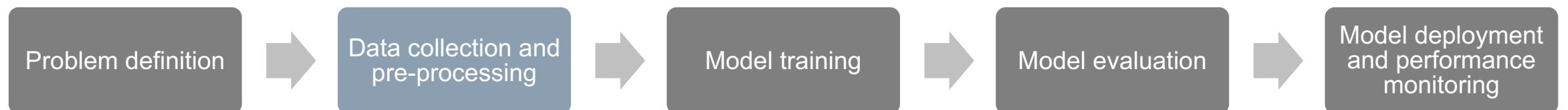
- Remove missing values and outliers, apply dimensionality reduction, normalize, rescale, ...



Step 2. Data collection and pre-processing

Finally, and before training any machine learning algorithm, we perform **data segregation**.

- We **separate the target** value from the input features
- We split the data collection into
 - Training set: used to fit our model parameters
 - Validation set: used to have an unbiased estimation of the model generalization capabilities during training, and tune hyperparameters of our model
 - Test set: used to evaluate performance of a final version of the model

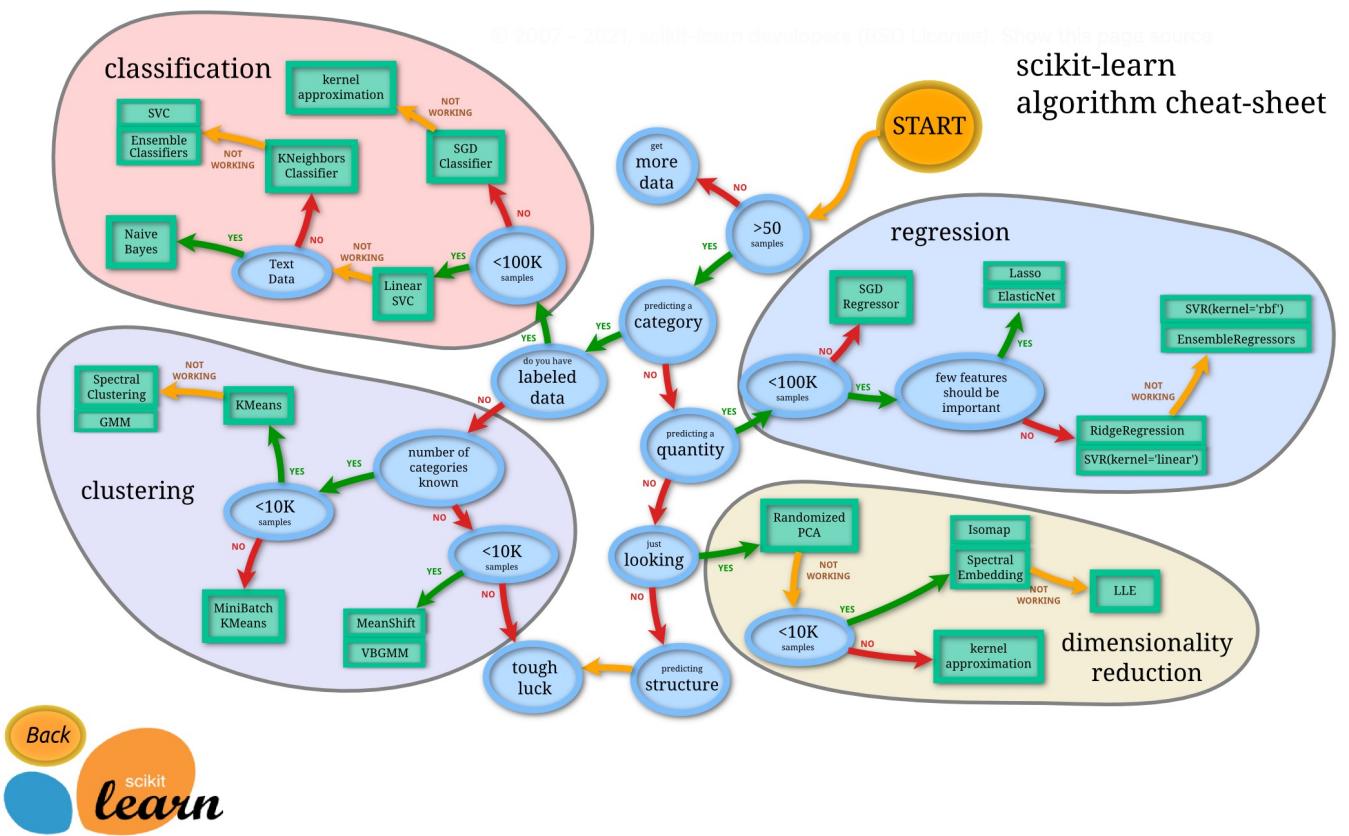


Step 3. Model training

We need to **select an algorithm** to be trained on our data.

- E.g., the prediction of a machine failure can be defined as a classification or a regression problem

To find out which algorithm is the best for our data set we have to test them.



https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

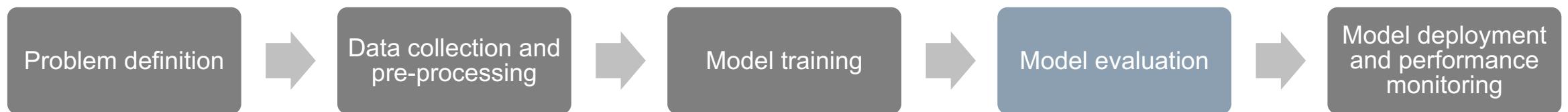


Step 4. Model evaluation

Training and evaluation of the model are iterative processes:

- First, we train our model with the training set
- Then evaluate its performance with validation set with evaluation metrics
- Based on this information, we **tune our algorithm's hyperparameters**

This iterative process continues unless we decide that we can't improve our algorithm anymore.



Step 4. Model evaluation

Training and evaluation of the model are iterative processes:

- First, we train our model with the training set
- Then evaluate its performance with validation set with evaluation metrics
- Based on this information, we **tune our algorithm's hyperparameters**

This iterative process continues unless we decide that we can't improve our algorithm anymore.

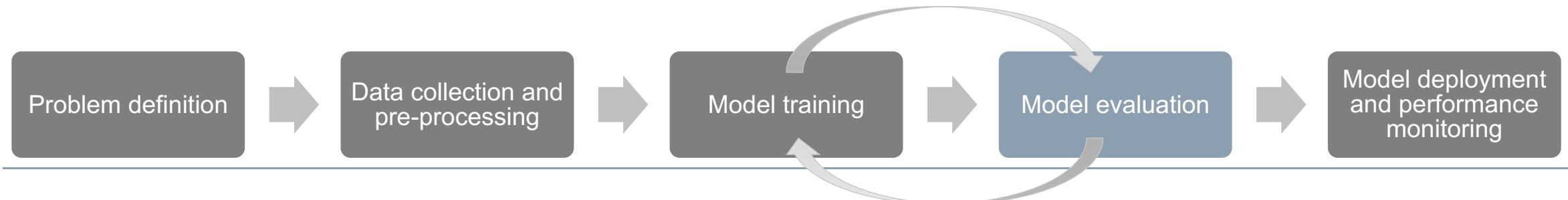


Step 4. Model evaluation

Training and evaluation of the model are iterative processes:

- First, we train our model with the training set
- Then evaluate its performance with validation set with evaluation metrics
- Based on this information, we **tune our algorithm's hyperparameters**

This iterative process continues unless we decide that we can't improve our algorithm anymore.



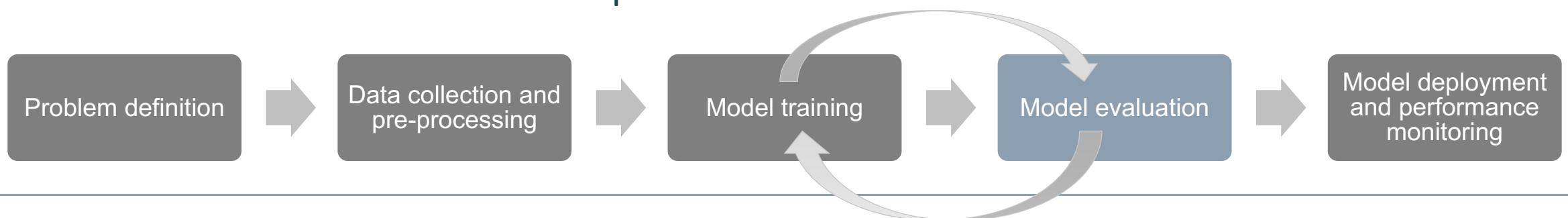
Step 4. Model evaluation

Training and evaluation of the model are iterative processes:

- First, we train our model with the training set
- Then evaluate its performance with validation set with evaluation metrics
- Based on this information, we **tune our algorithm's hyperparameters**

This iterative process continues unless we decide that we can't improve our algorithm anymore.

Then we **use the test set** to see the performance on **unknown data**.



Classification models evaluation – the confusion metrix

Confusion Matrix describes the performance of the model.

There are 4 important terms:

True Positives: The cases in which we predicted YES, and the actual output was also YES

True Negatives: The cases in which we predicted NO, and the actual output was NO

False Positives: The cases in which we predicted YES, and the actual output was NO

False Negatives: The cases in which we predicted NO, and the actual output was YES

Confusion Matrix

n=165	Predicted:NO	Predicted:YES
Actual: NO	50	10
Actual: YES	5	100

Classification Accuracy is what we usually mean, when we use the term accuracy. It is the ratio of number of correct predictions to the total number of input samples.

$$\text{Accuracy} = (150/165) = 0.909$$

Regression models evaluation – metrics

Mean Absolute Error (MAE):

- Average difference between the original and predicted values
- Measure how far predictions were from the actual output
- Does not give and idea about the direction of error.

$$\text{Mean Absolute Error} = \frac{1}{N} \sum_{j=1}^N |y_j - \hat{y}_j|$$

Mean Squared Error (MSE):

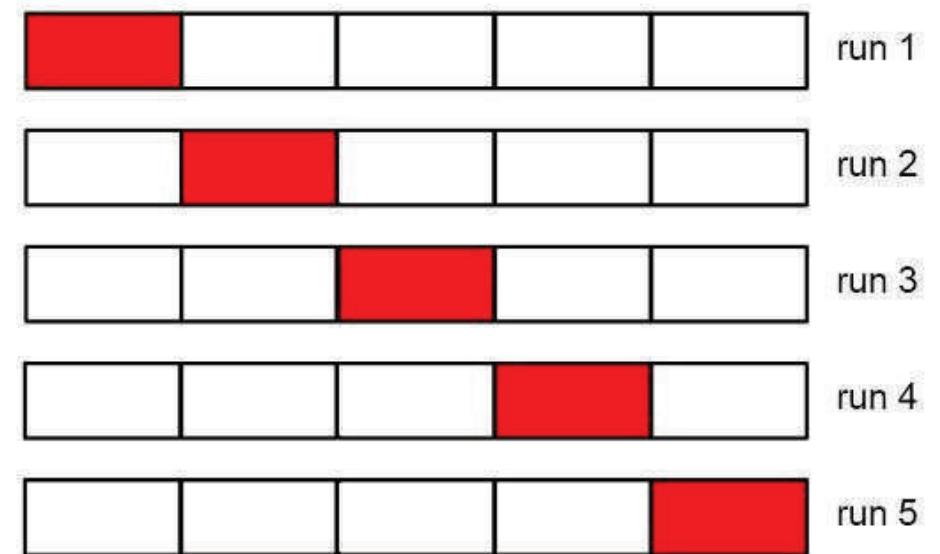
- Similar to MAE
- It takes the average of the square of the difference between original and predicted value
- Larger errors become more pronounced, so that the model can focus on larger errors.

$$\text{Mean Squared Error} = \frac{1}{N} \sum_{j=1}^N (y_j - \hat{y}_j)^2$$

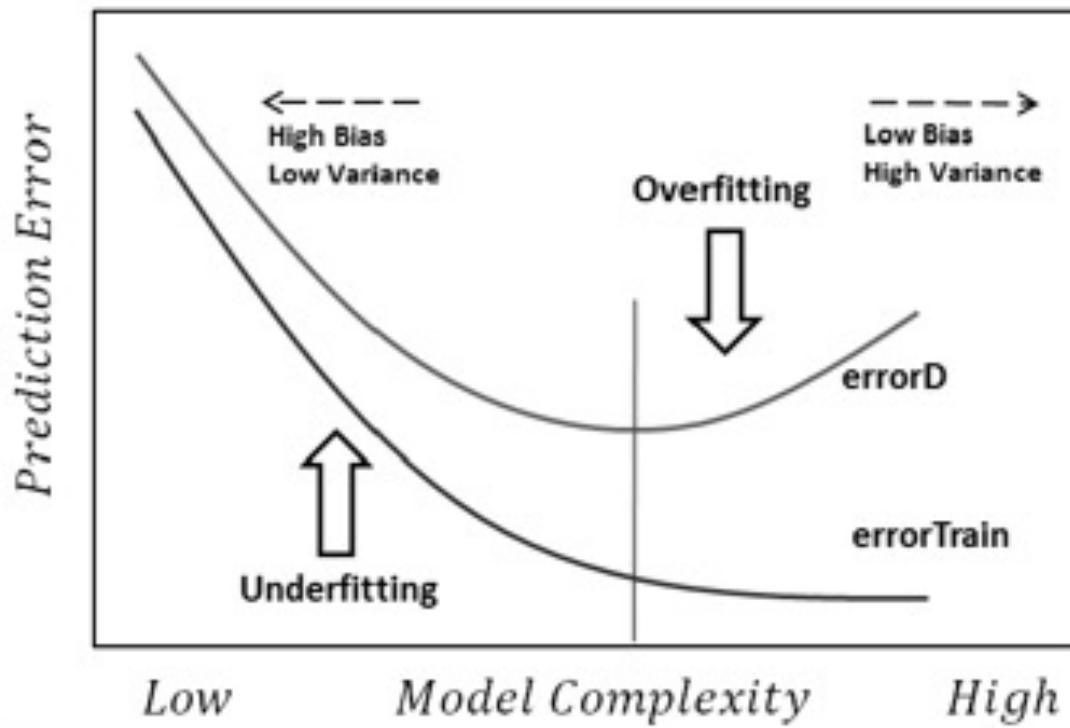
Cross validation

Cross-validation is a resampling method to iteratively use different portions of the dataset to train and validate a model among iterations.

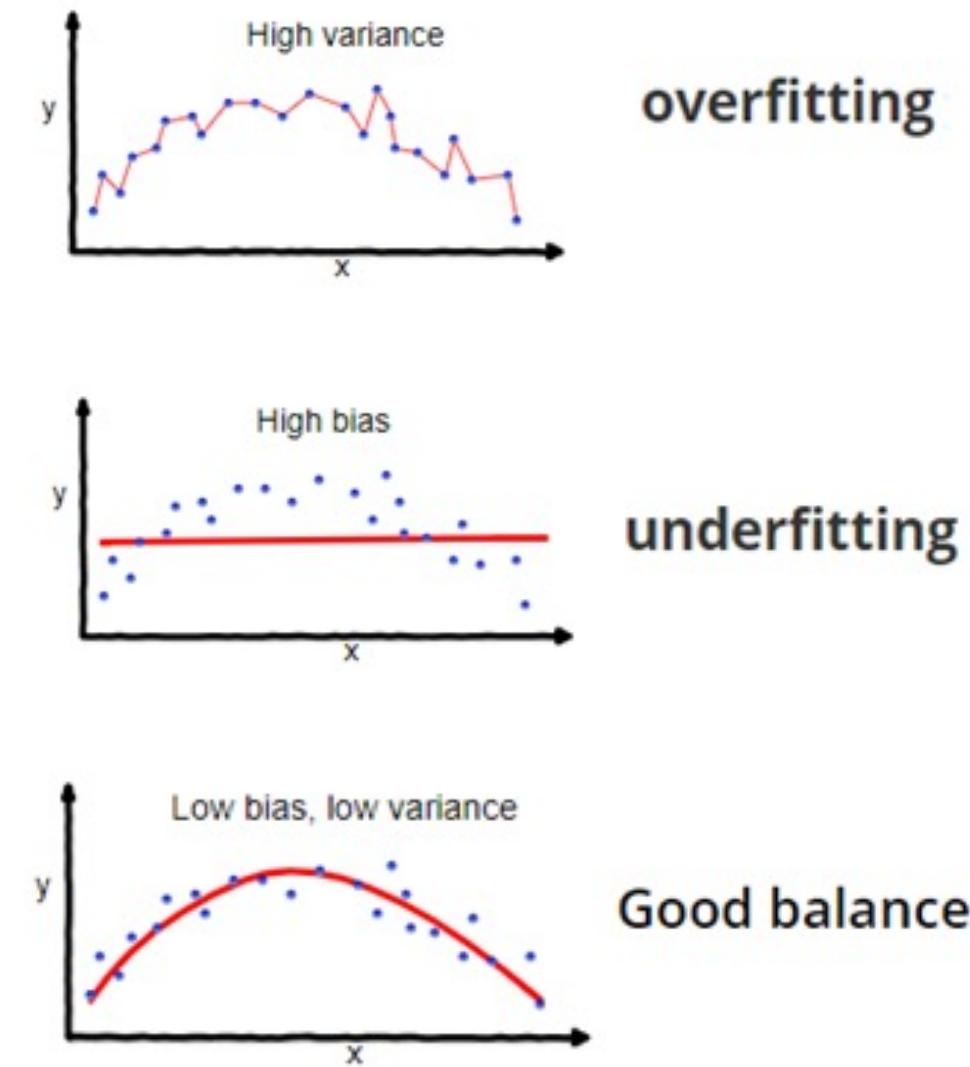
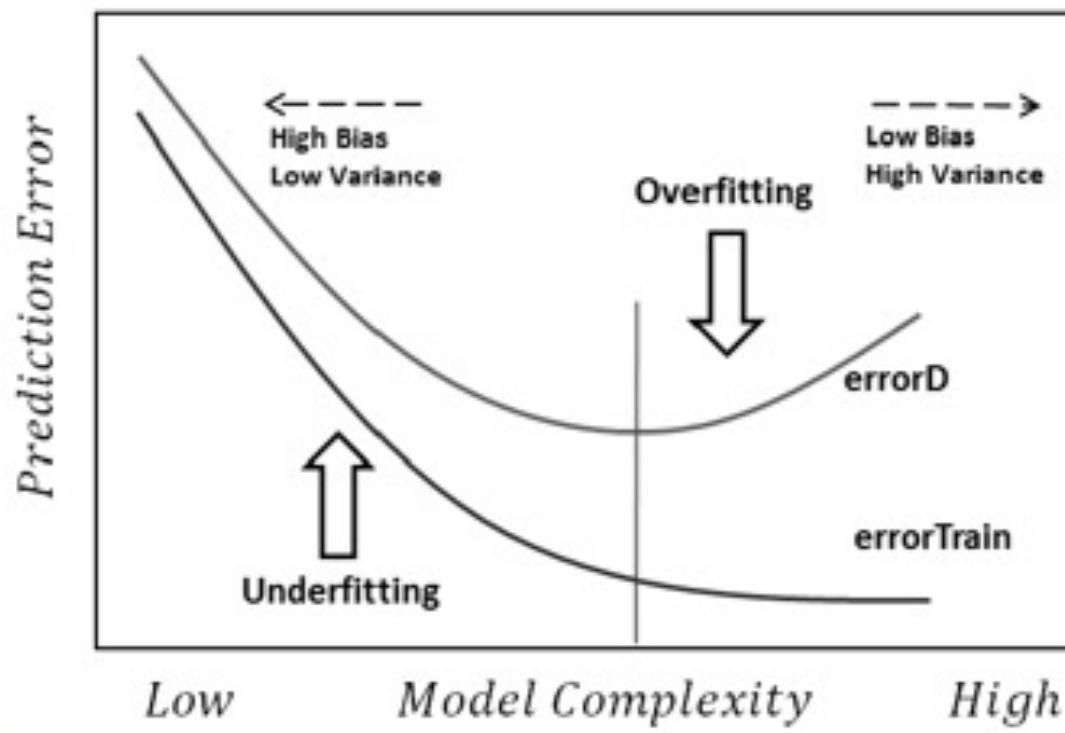
- Particularly useful when the dataset size is small
- It can be also used for hyper-parameters selection



Underfitting and overfitting



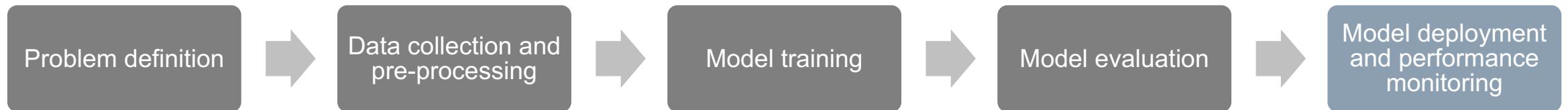
Underfitting and overfitting



Step 5. Model deployment and performance monitoring

After a successful training, we can **deploy the model**. Here we have often to consider the following issues:

- Real time requirements
- Robust hardware (sensors and processor)



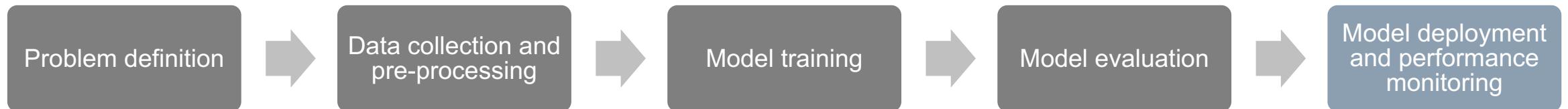
Step 5. Model deployment and performance monitoring

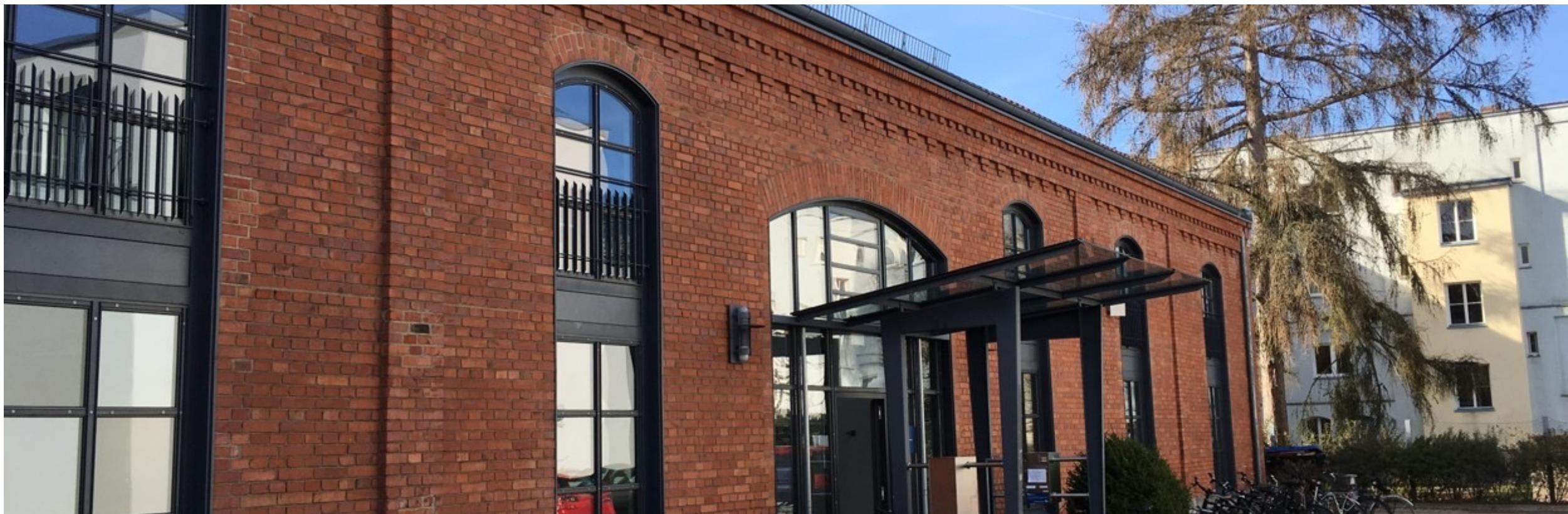
After a successful training, we can **deploy the model**. Here we have often to consider the following issues:

- Real time requirements
- Robust hardware (sensors and processor)

We must **monitor the performance** of our model and make sure it still produces satisfactory results.

- Sensors can malfunction and provide wrong data
- The data can be out of the trained range of the model





Time serie fundamentals

Common ML tasks with time series



The machine learning tasks

Machine learning (ML) can be regarded as a collection of methods that enables us to solve **tasks** which would be too difficult to be solved by a fixed written program designed by human beings.

From a phylosophycal point of view this is interesting because it can be seen as an attempt to formalize the concept of **intelligence**.

ML usually describes how machines should process **examples**.

- Examples are collections of **features**
- In the case of time series, features are the observations sorted in time.

Time series classification

Let $\mathcal{D} = \{(S^{(1)}, c^{(1)}), \dots, (S^{(N)}, c^{(N)})\}$ be a dataset of pairs, where

- $S^{(i)}$ is a time series
- and $c^{(i)} \in \{0,1\}^K$ denotes the one-hot encoded class vector (also said, labels vector).

$$f(\text{[time series plot]}) = \boxed{\text{[grey circle]} \text{ [yellow circle]}}$$

$$f(\text{[time series plot]}) = \boxed{\text{[white circle]} \text{ [yellow circle]}}$$

Then, a time series classification task is about learning a mapping function f , such that:

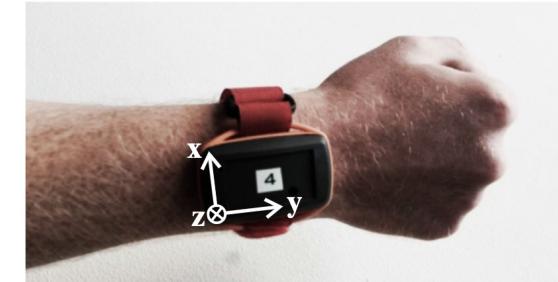
$$f(S^{(i)}) = c^{(i)}, \forall i \in \{1, \dots, N\}$$

Example of time series classification

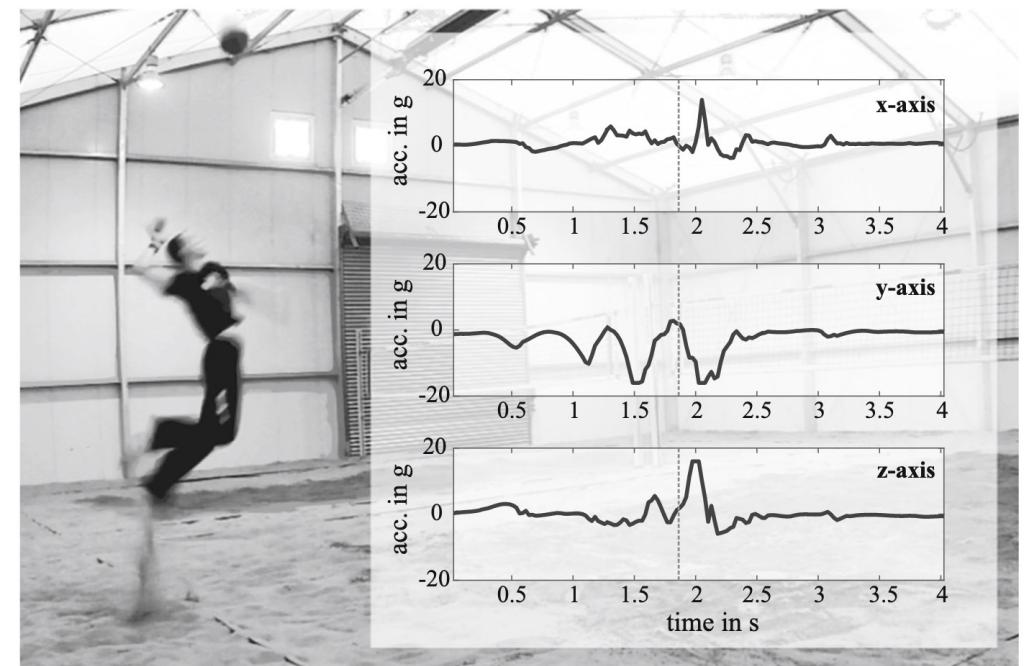
- Monitoring of player actions could help identifying and understanding risk factors and prevent such injuries.

Actions:

- Underhand serve
- Overhand serve
- Jump serve
- Underarm set
- Overhead set
- Shot attack
- Spike
- Block
- Dig
- Null class.



Sensor attachment at the wrist of the dominant hand with a soft, thin wristband

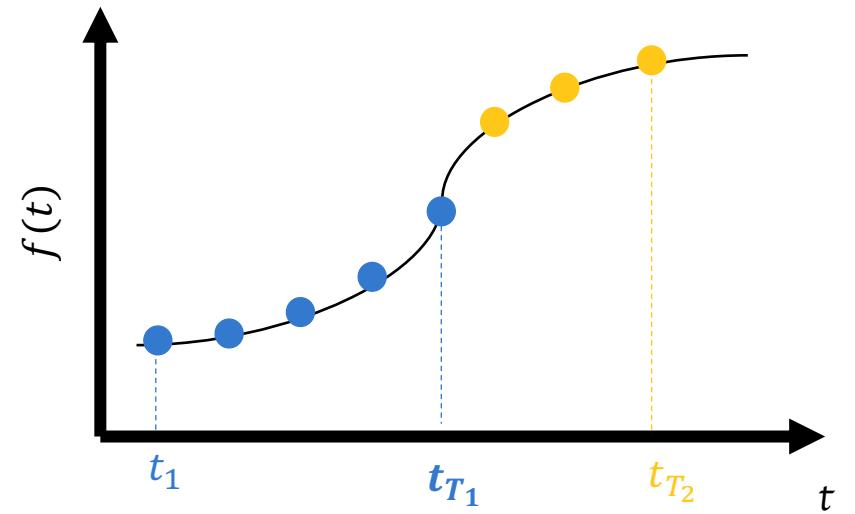


Time series forecasting

Let $S = \{s_1, \dots, s_{T_1}, s_{T_1+1}, \dots, s_{T_2}\}$ be a time series, with s_i being the i -th observation collected at time t_i , and $t_i < t_j, \forall j$.

Then, a time series forecasting task is about predicting future values of a time series given some past data, i.e.,

$$f(s_1, \dots, s_{T_1}) = (s_{T_1+1}, \dots, s_{T_2})$$



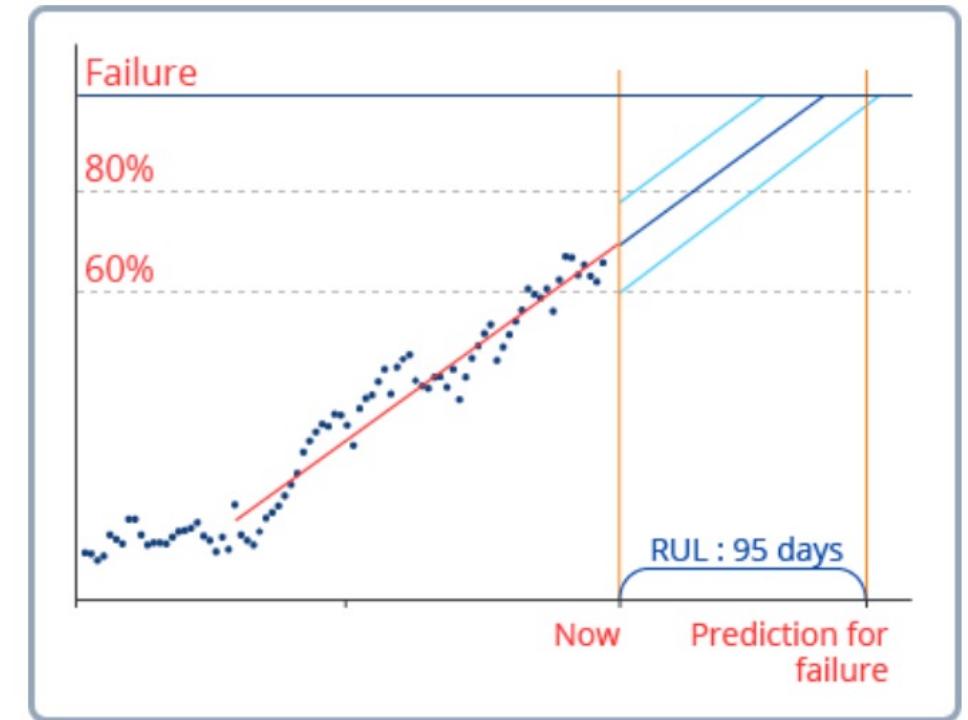
Example of time series forecasting: Prediction of Remaining Useful Life (RUL)

Objective:

- Machine components **degrade over time**
- This causes malfunctions in the production line
- Replace component **before** the malfunction!

Realization:

- Use data of malfunctioning systems
- Fit a regression model to the data and predict the RUL
- Use the model on active systems and apply necessary maintenance



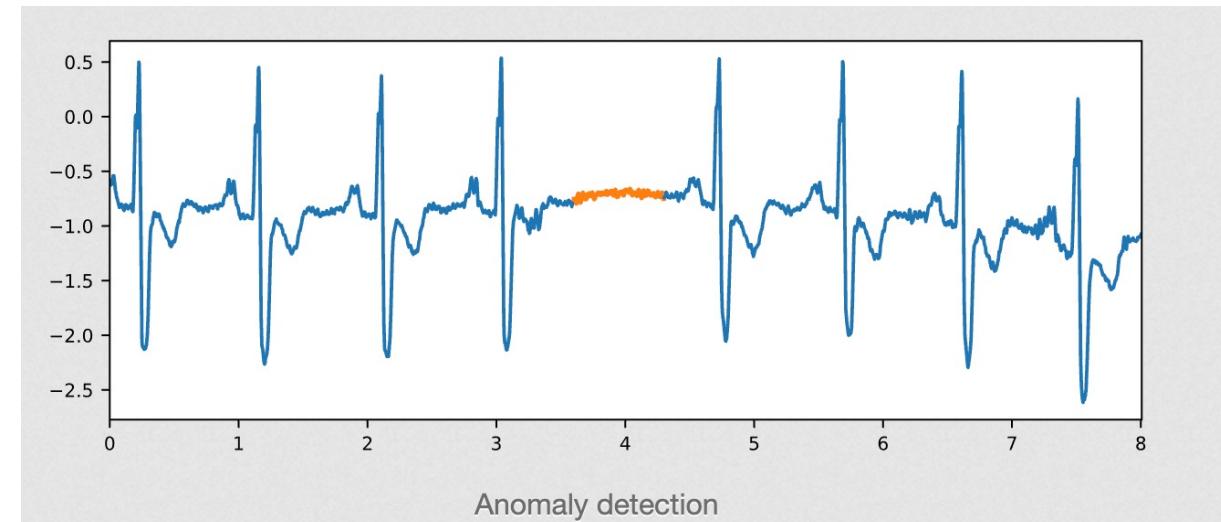
Anomaly detection on time series

Let $S = \{s_1, \dots, s_T\}$ be a time series, with s_i being the i -th observation collected at time t_i .

Then, an anomaly detection task is that of predicting the probability of a certain observation to be anomalous,

$$f(s_i) = p_i, \forall i \in \{1, \dots, T\}$$

with $p_i = 0$ for regular data and $p_i = 1$ for anomalous data.



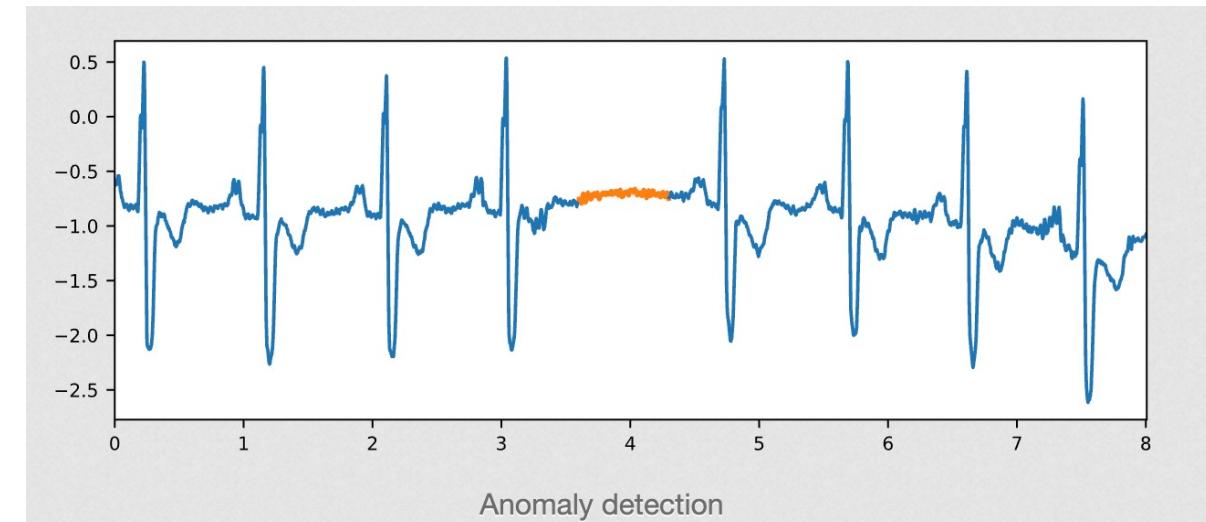
(a) <https://siebert-julien.github.io/time-series-analysis-python/>

Examples of anomaly detection on time series

Anomaly detection, sometimes also called outliers detection or novelty detection, is therefore the *task of finding abnormal data points* (equiv., outliers).^(a)

Examples of real world applications of anomaly detection on time series are:

- detecting fraud transactions
- fraudulent insurance claims
- cyber attacks to detecting abnormal equipment behaviors



^(a) <https://siebert-julien.github.io/time-series-analysis-python/>

Time series segmentation

Let $S = \{s_1, \dots, s_T\}$ be a time series, with s_i being the i -th observation collected at time t_i .

Time series segmentation is the task of splitting data points into segments, which reveal underlying properties of the generation process, which can formalized as the process of assigning every sample to its corresponding cluster, i.e., $f(s_i) = c_i$, with $c_i \in \{c_1, \dots, c_M\}$.

Example of time series segmentation

A typical example is that of online handwriting recognition.

A time series describes a list of coordinates, e.g., the point of a pen over a touchscreen surface, collected over time.

The task is to determine segments corresponding to a single letter.

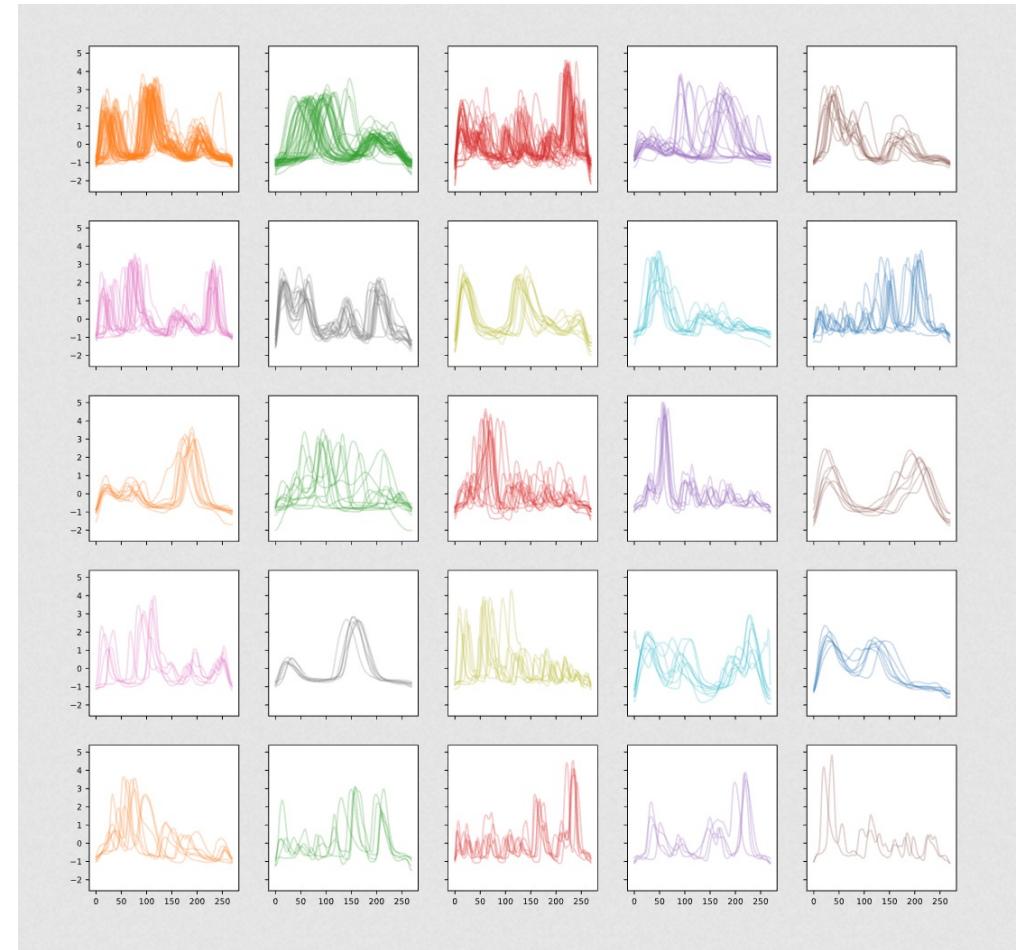


Time series clustering

Clustering can be applied to time series with the goal of grouping together similar sequences.

This finds important application in data analysis and pre-processing

- Find similar customers behaviours and exploit this information in recommender systems



(a) <https://siebert-julien.github.io/time-series-analysis-python/>



Time series fundamentals

Example: Linear regression for time series forecasting



Motivation

Linear regression is used in multiple scenarios each and every day!

- E.g., Trend Analysis in financial markets, sales and business



Motivation

Linear regression is used in multiple scenarios each and every day!

- E.g., Trend Analysis in financial markets, sales and business

The problem has to be simple:

- Dataset is small
- Linear model is enough, i.e., Trend Analysis
- Linear models are the basis for complex models, i.e., Deep Networks



Linear Regression is a method to fit **linear models** to our data!

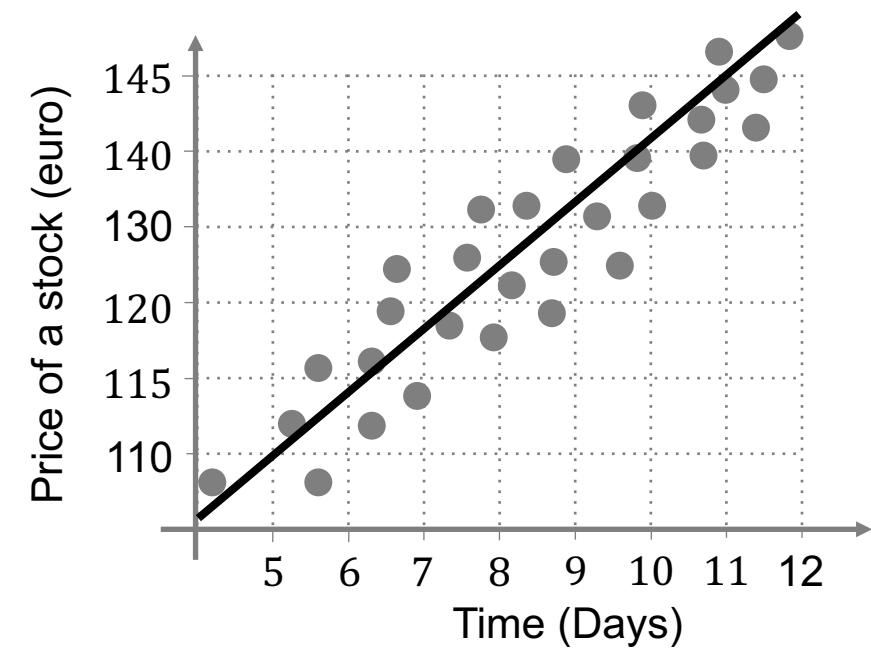
The linear model:

$$f(\mathbf{x}) = w_0 \cdot x_0 + w_1 \cdot x_1 = \mathbf{y}$$

$$\mathbf{x} = \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} = \begin{pmatrix} 1 \\ \text{Days} \end{pmatrix}$$

y = Stock price

w = Weights



Linear Regression is a method to fit **linear models** to our data!

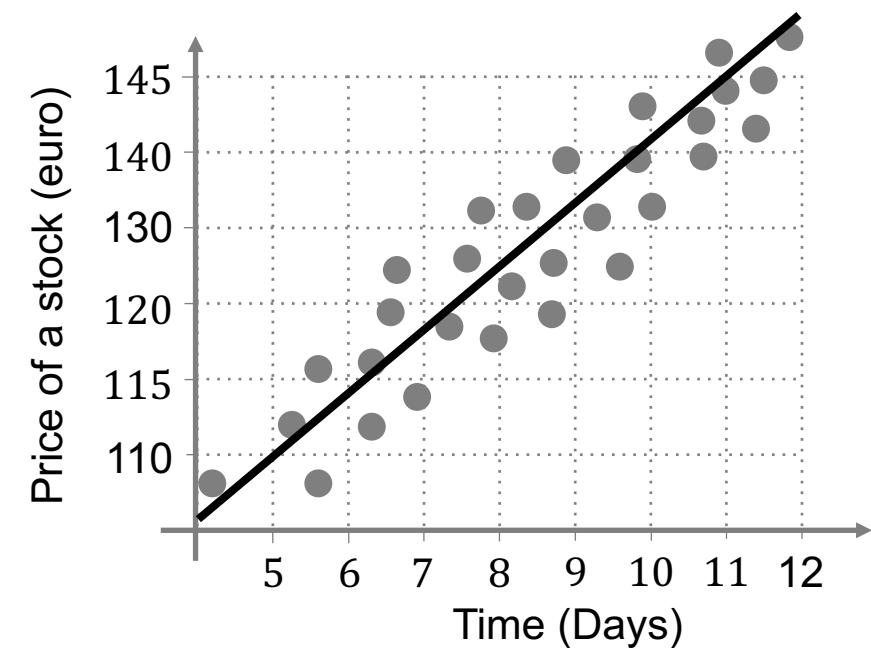
The linear model:

$$f(\mathbf{x}) = w_0 \cdot x_0 + w_1 \cdot x_1 = y$$

The linear model (in our example):

$$f(\mathbf{x}) = 70 \cdot x_0 + 6.5 \cdot x_1 = y$$

→ Finding a good w_0 and w_1 is called **fitting**!



The previous description of the linear model
is not accurate!

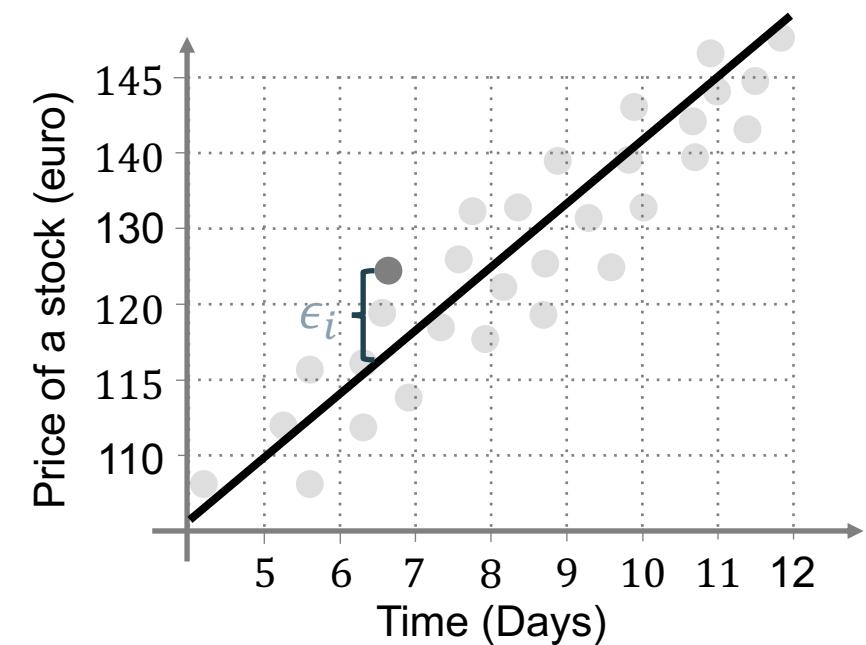
Reason: Real Systems produce noise!

Linear model with noise:

$$f(\mathbf{x}) = w_0 \cdot x_0 + w_1 \cdot x_1 + \epsilon_i$$

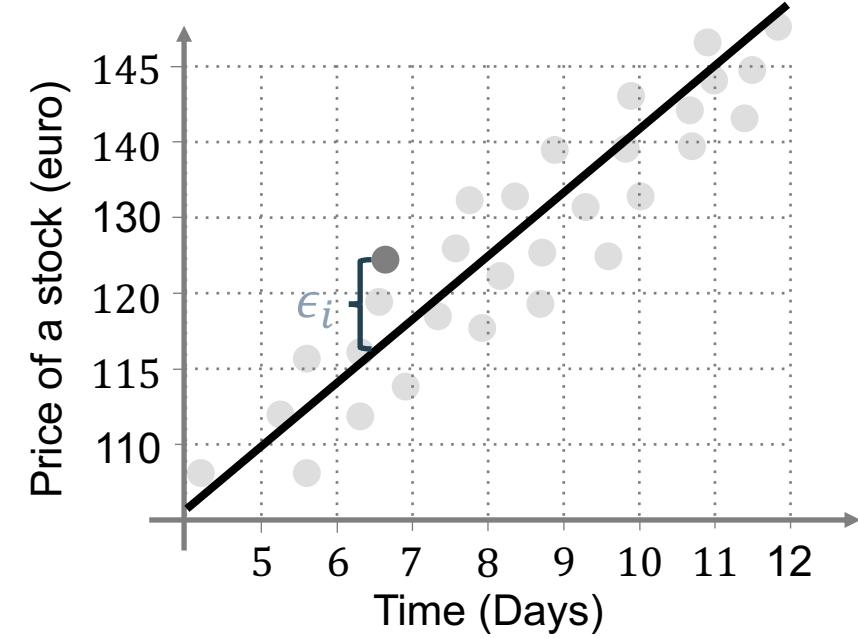
The ϵ_i and the summation ϵ of
all samples is called **Residual Error**!

Typically, we assume ϵ is Gaussian
Distributed (i.e. Gaussian noise)



A more general formulation:

$$f(\mathbf{x}) = \sum_{j=1}^D w_j x_j + \epsilon = \mathbf{y}$$

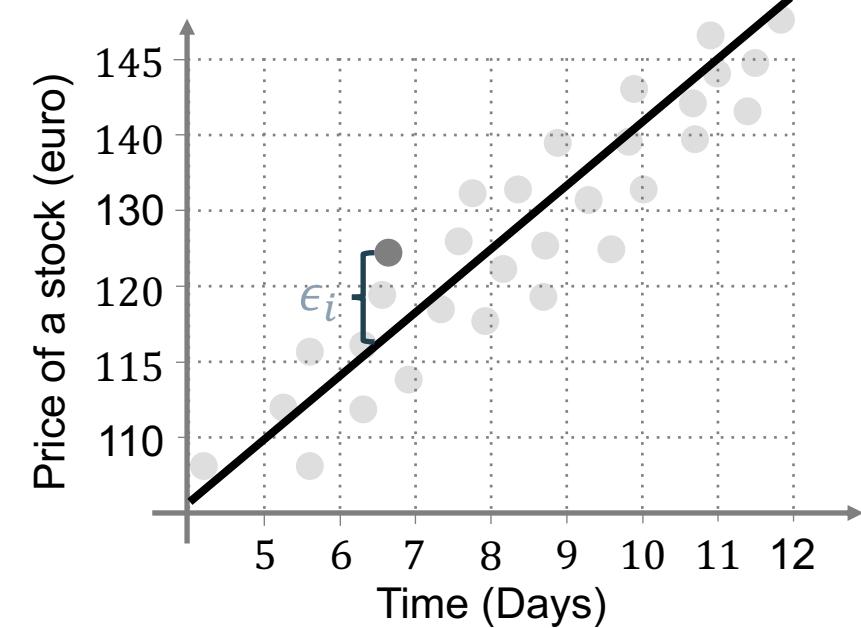


A more general formulation:

$$f(\mathbf{x}) = \sum_{j=1}^D w_j x_j + \epsilon = \mathbf{y}$$

The **model parameters** are:

$$\theta = \{w_0, \dots, w_n, \sigma\} = \{\mathbf{w}, \sigma\}$$



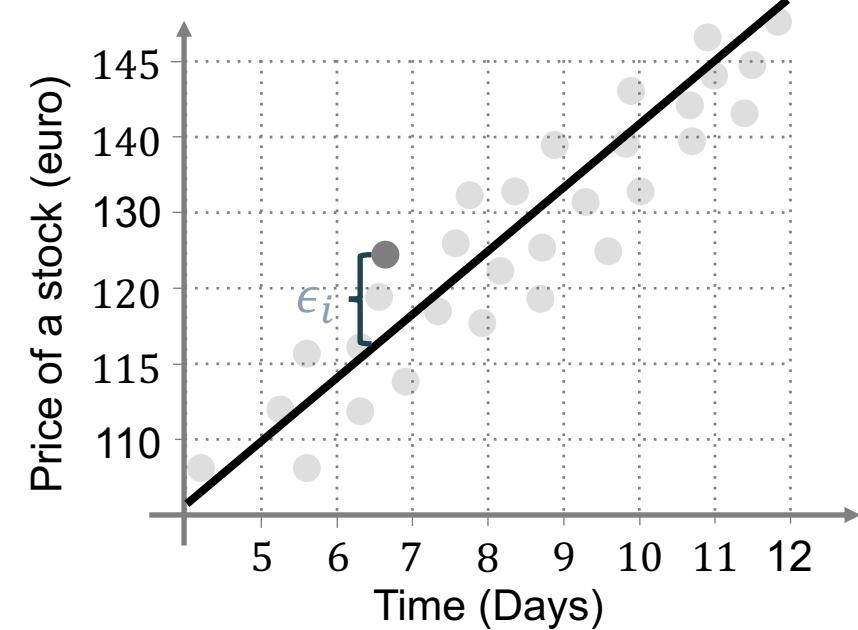
A more general formulation:

$$f(\mathbf{x}) = \sum_{j=1}^D w_j x_j + \epsilon = \mathbf{y}$$

The **model parameters** are:

$$\boldsymbol{\theta} = \{w_0, \dots, w_n, \sigma\} = \{\mathbf{w}, \sigma\}$$

We define the **optimal parameters** as
(Maximum likelihood estimation, MLE):



A solution to this problem is given by the minimization of the negative log likelihood:

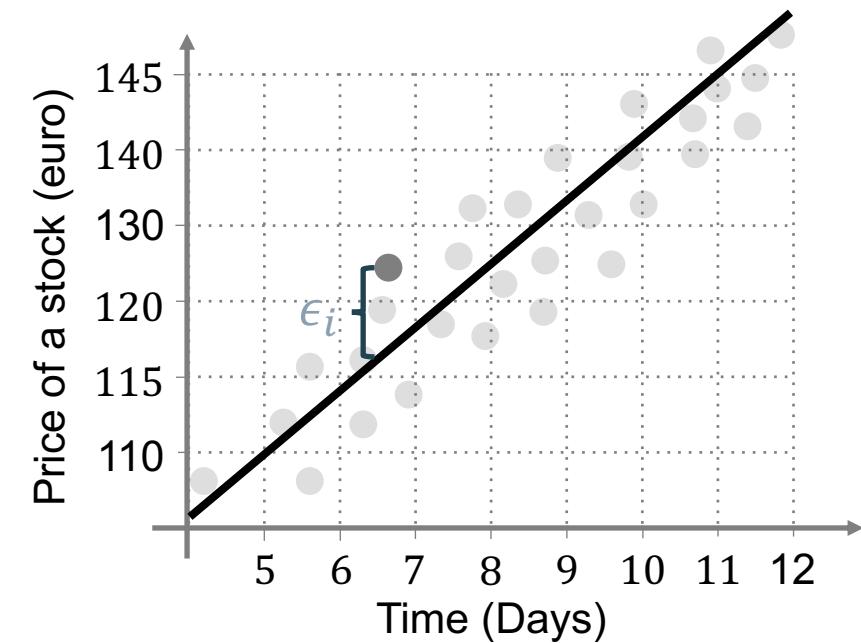
$$\text{NLL}(\theta) = \frac{1}{2} (\mathbf{y} - \mathbf{x}\mathbf{w})^T (\mathbf{y} - \mathbf{x}\mathbf{w})$$

We find the minimum conventionally by using **the derivative**:

$$\text{NLL}'(\theta) = \mathbf{x}^T \mathbf{x}\mathbf{w} - \mathbf{x}^T \mathbf{y}$$

The solution, i.e., the minimum is:

$$\mathbf{w} = (\mathbf{x}^T \mathbf{x})^{-1} \mathbf{x}^T \mathbf{y}$$





Time series fundamentals

Recap



Recap

Types of machine learning

- Supervised learning
- Unsupervised learning
- Reinforcement learning

ML pipeline

- From problem definition to model deployment
- Training and Evaluation good practices

Common ML tasks with time series

- Time series classification
- Forecasting
- Anomaly detection
- Time series segmentation
- Time series clustering

Linear regression for time series forecasting



Machine Learning for Time Series

(MLTS or MLTS-Deluxe Lectures)

Dr. Dario Zanca

Machine Learning and Data Analytics (MaD) Lab
Friedrich-Alexander-Universität Erlangen-Nürnberg
25.10.2022

Organisational Information

Lectures (online)

A new lecture recording every Tuesday.

Consultation hours from November 15th, h. 8:15 – 9:30

Exercises (online)

Live Zoom Session starting on November 9th

Recordings will be uploaded

Project (online)

Introduction during the first exercise Live Zoom Session (November 9th)

Applications started on Nov 9th

-
- Time series fundamentals and definitions (2 lectures)
 - Bayesian Inference (1 lecture) ←
 - Gaussian processes (2 lectures)
 - State space models (2 lectures)
 - Autoregressive models (1 lecture)
 - Data mining on time series (1 lecture)
 - Deep learning on time series (4 lectures)
 - Domain adaptation (1 lecture)

In this lecture...

1. Bayes Theorem
2. Bayesian Model Selection
3. Prior Distributions
4. Linear Regression (Bayesian treatment)



Bayesian Inference

Bayes' Theorem



The **Bayes' Theorem** was formulated by the English philosopher **Thomas Bayes** (1701 – 1761), whose notes were edited and published posthumously by Richard Price.

Bayes' theorem is stated mathematically as the following equation:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

where A and B are events, and $P(B) \neq 0$.

Portrait from: Terence O'Donnell, *History of Life Insurance in Its Formative Years* (Chicago: American Conservation Co., 1936), p. 335





Posterior probability

The probability of event A occurring given that B is true

Likelihood

The probability of event B occurring given that A is true

$$P(A|B) = \frac{P(B|A) P(A)}{P(B)}$$

Marginal probability

The probability of observing B without any given conditions

Prior probability

The probability of observing A without any given conditions

Portrait from: Terence O'Donnell, *History of Life Insurance in Its Formative Years* (Chicago: American Conservation Co., 1936), p. 335

Bayes' Theorem

An example. Iterative application of the Bayes' theorem.

We know that:

- Disease chance: 1%
- Test accuracy: 95%

A: Having the disease

B: Testing positive to the disease

$P(B|A)$: Test accuracy (Likelihood)

$P(B)$: Prob. of a positive test (Marginal)

$P(A)$: Disease chance (Prior)

First positive test

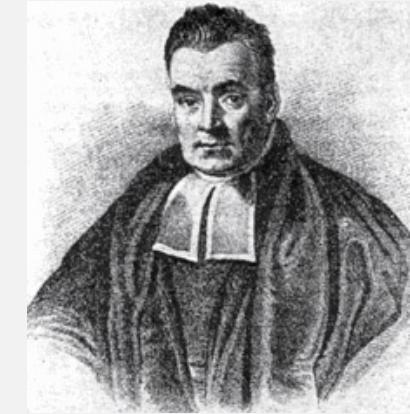
$$P(A|B) = \frac{.95 \times 0.01}{0.95 \cdot 0.01 + (1 - 0.95)(1 - 0.01)} = 0.161$$

Second positive test

$$P(A|B) = \frac{.95 \times 0.161}{0.95 \cdot 0.161 + (1 - 0.95)(1 - 0.161)} = 0.785$$

Third positive test

$$P(A|B) = \frac{.95 \times 0.785}{0.95 \cdot 0.785 + (1 - 0.95)(1 - 0.785)} = 0.986$$



$$P(A|B) = \frac{P(B|A) P(A)}{P(B)}$$

Portrait from: Terence O'Donnell, *History of Life Insurance in Its Formative Years* (Chicago: American Conservation Co., 1936), p. 335

Let \mathcal{D} denote the **observed data**,

$$\mathcal{D} = \{x^{(n)}, y^{(n)}\}$$

with $x^{(n)} \in \mathcal{R}$ represents the input, and $y^{(n)} \in \mathcal{R}$ represents the output (labels).

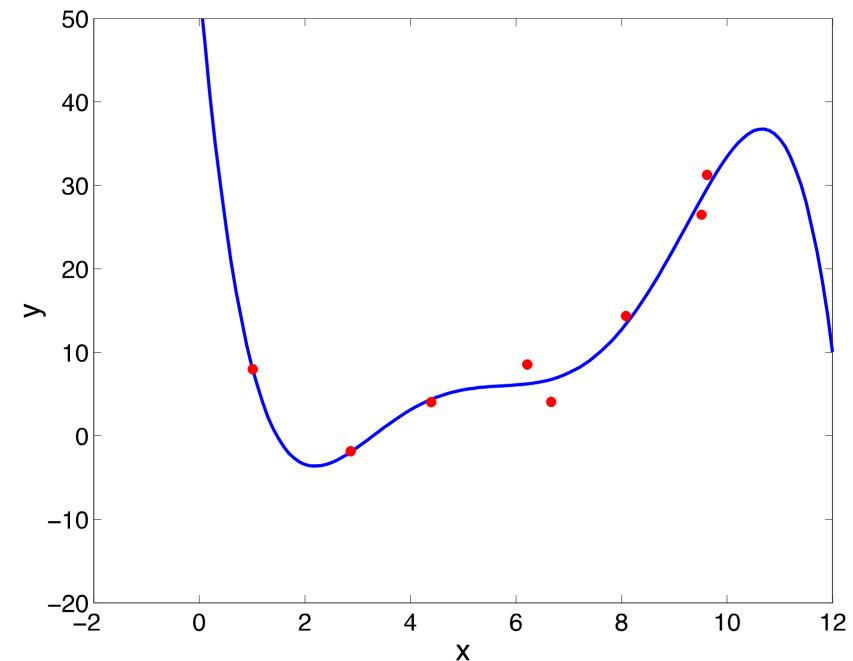
The **model** is defined as

$$y^{(n)} = \omega_0 + \omega_1 x^{(n)} + \omega_2 x^{(n)} \dots + \omega_m x^{(n)} + \epsilon$$

where data noise is Gaussian distributed, i.e., $\epsilon \sim \mathcal{N}(0, \sigma^2)$.

We denote with θ the **unknown parameters**,

$$\theta = (\omega_0, \dots, \omega_m, \sigma)$$

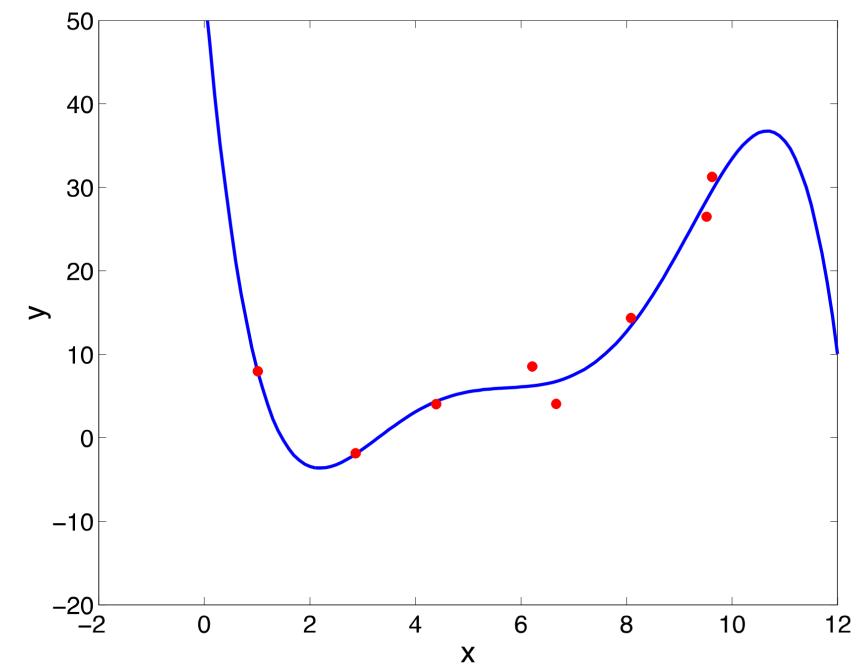


Data: $\mathcal{D} = \{x^{(n)}, y^{(n)}\}$

Model: $y^{(n)} = \omega_0 + \omega_1 x^{(n)} + \omega_2 x^{(n)} \dots + \omega_m x^{(n)} + \epsilon$

Unknown parameters: $\theta = (\omega_0, \dots, \omega_m, \sigma)$

Goal: To infer θ from the data and to predict future outputs $p(y|x, \theta, \mathcal{D})$



$p(\mathcal{D}|\theta)$: likelihood of θ

$p(\theta)$: prior probability of θ

$p(\theta|\mathcal{D})$: posterior of θ given \mathcal{D}

$p(\mathcal{D})$: marginal probability of \mathcal{D}

$p(y|x, \mathcal{D})$: predictive distribution

$p(\mathcal{D}|\theta)$: likelihood of θ

$p(\theta)$: prior probability of θ

$p(\theta|\mathcal{D})$: posterior of θ , given \mathcal{D}

$p(\mathcal{D})$: marginal probability of \mathcal{D}

$p(y|x, \mathcal{D})$: predictive distribution

Bayes' rule:

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{P(\mathcal{D})}$$

Prediction of a new point:

$$p(y|x, \mathcal{D}) = \int p(y|\theta, x, \mathcal{D})p(\theta|\mathcal{D}) d\theta$$

- In contrast to the maximum likelihood estimation (MLE), in Bayesian learning **we average over possible parameter settings** rather than optimizing over parameter space.
- Bayesian inference gives us a **systematic way to express our uncertainty** about future predictions. Prediction is not just a point estimate (as for MLE) but has a probability form that expresses the uncertainty about the predictions.



Bayesian Inference

Bayesian Model Selection





The **principle of Occam's razor** in its original formulation states that:

"**Entia non sunt multiplicanda praeter necessitatem**"

(In English, "Entities should not be multiplied unnecessarily")

Many scientists have adopted or reformulated the Occam's Razor principle, which is often cited in stronger forms, as in the following statement:

- "If you have two theories that both explain the observed facts, then you should use the simplest until more evidence comes along"
- "One should pick the simplest model that adequately explains the data"



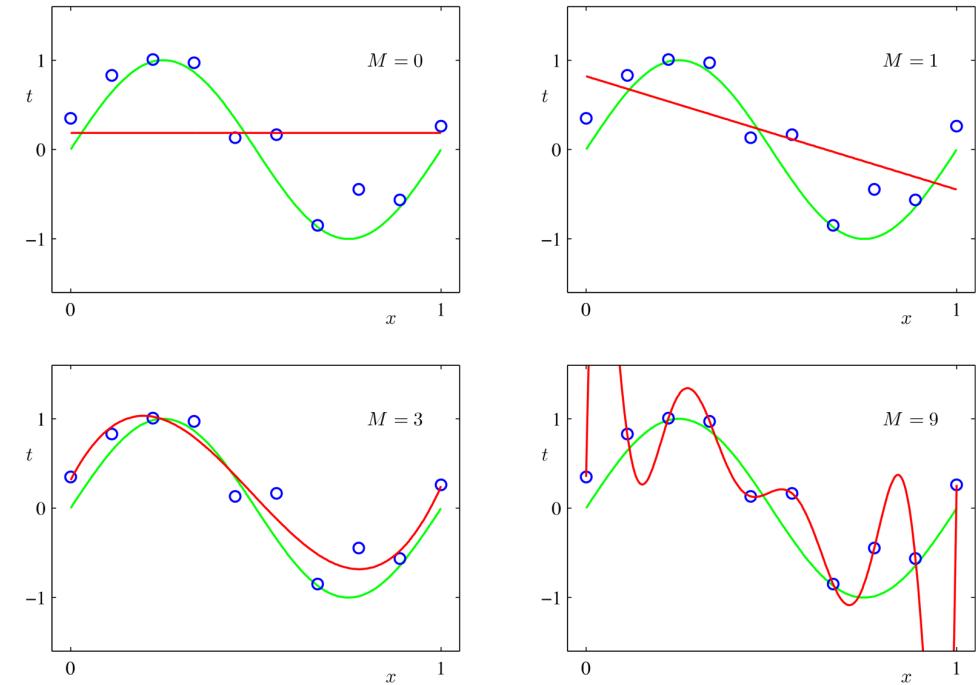
Picture from:
<https://www.britannica.com/topic/Occams-razor>

The model Selection problem

We could perform K-fold cross-validation (CV) to estimate the generalization error of all candidates model.

However, it requires fitting each candidate model K times!

→ A more efficient approach is given by Bayesian modelling



Which of the above models represents data the best?

We can compare different models using the marginal likelihood:

$$p(\mathcal{D}|M_i) = \int p(\mathcal{D}|\theta, M_i)p(\theta|M_i) d\theta$$

- Model classes that are **too simple** are unlikely to generate the data set.
- Model classes that are **too complex** can generate many possible data sets, so again, they are unlikely to generate that particular data set \mathcal{D} .

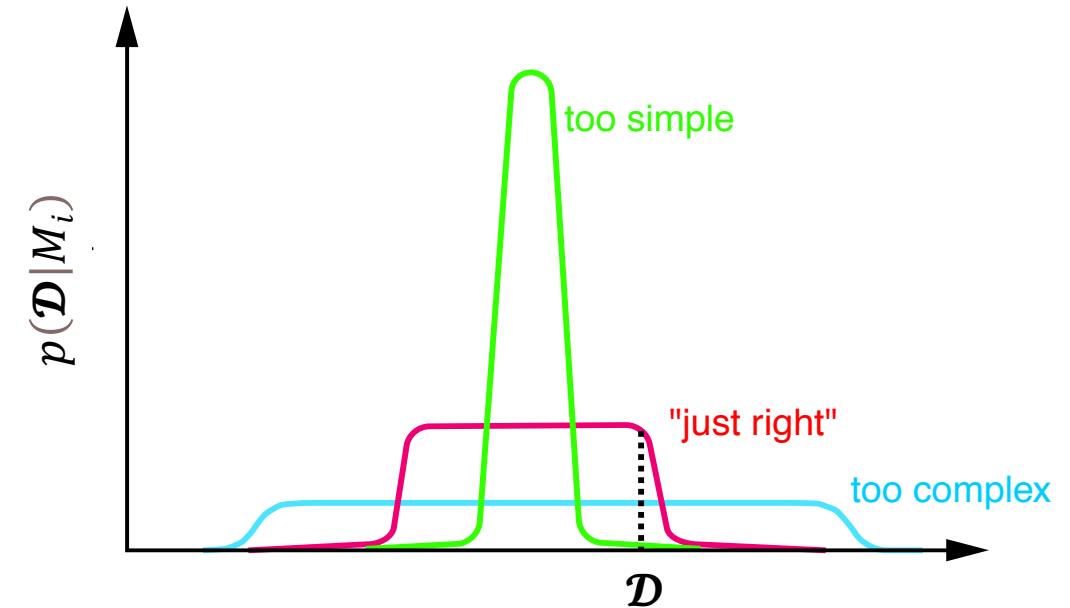


Image from: Rasmussen, C., & Ghahramani, Z. (2000). Occam's razor. Advances in neural information processing systems, 13.

To understand the Bayesian Occam's razor, we notice that:

$$\int_{\mathcal{D}} p(\mathcal{D}|M_i) = 1$$

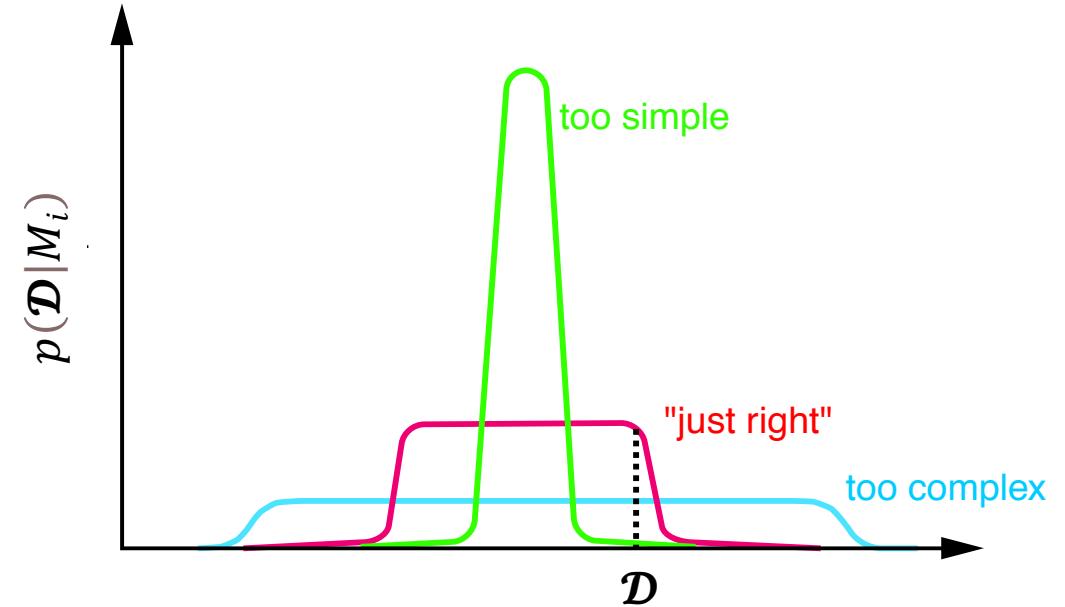


Image from: Rasmussen, C., & Ghahramani, Z. (2000). Occam's razor. Advances in neural information processing systems, 13.

To understand the Bayesian Occam's razor, we notice that:

$$\int_{\mathcal{D}} p(\mathcal{D}|M_i) = 1$$

Intuitively, complex models which can predict many datasets, must spread their probability mass
→ They don't attribute large probability for any given data set as simpler models.

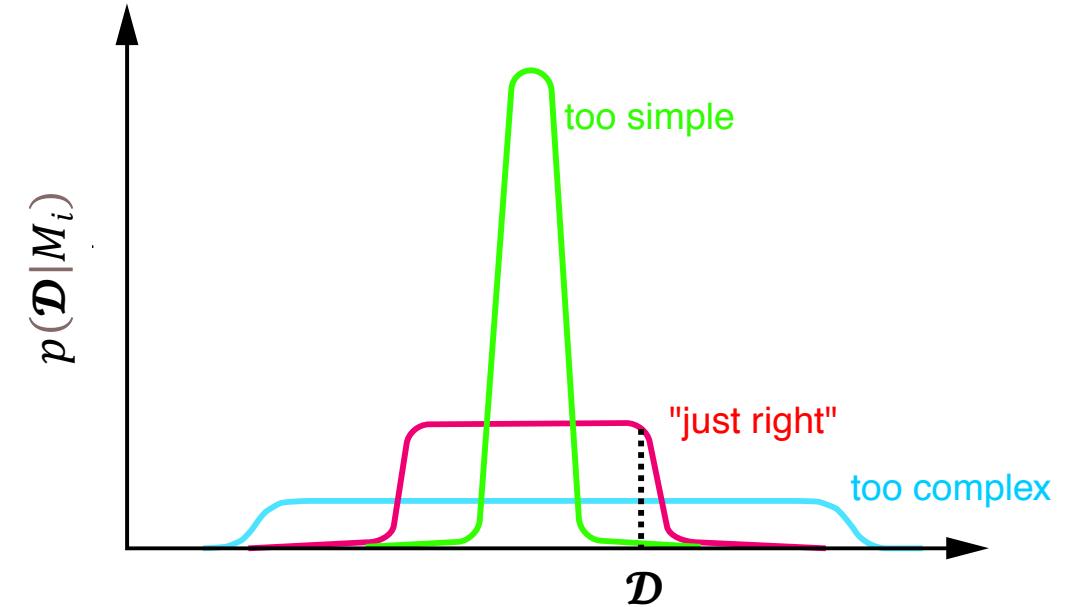


Image from: Rasmussen, C., & Ghahramani, Z. (2000). Occam's razor. Advances in neural information processing systems, 13.

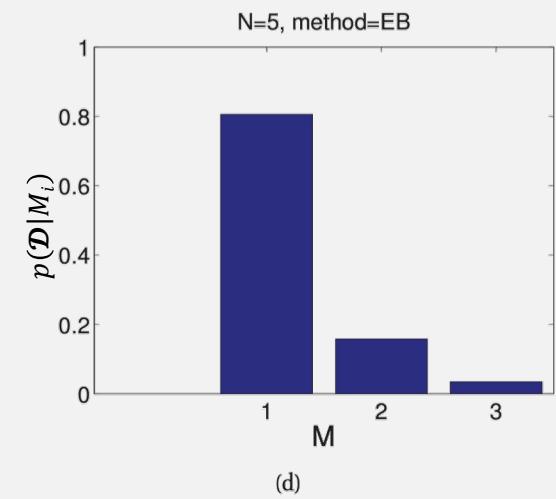
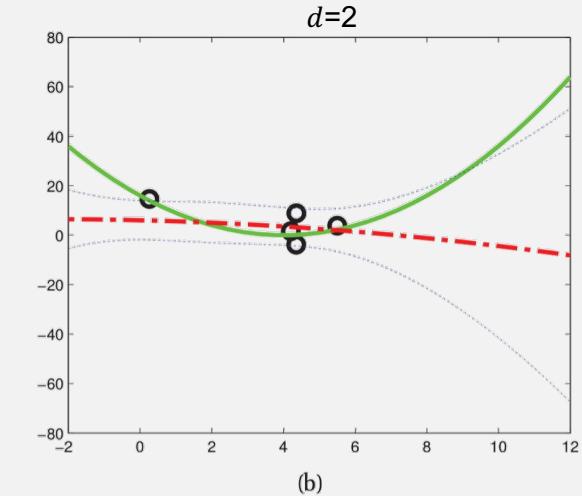
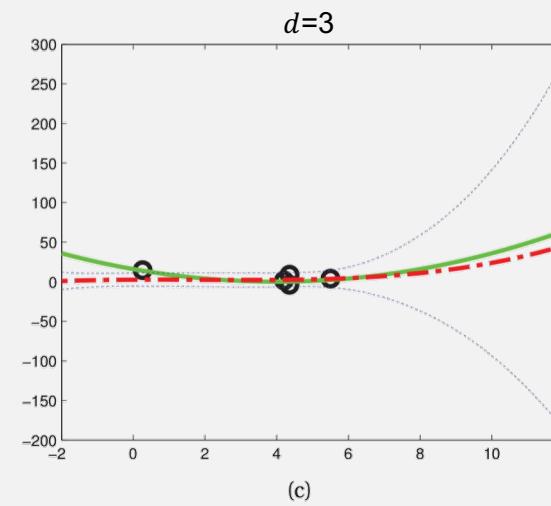
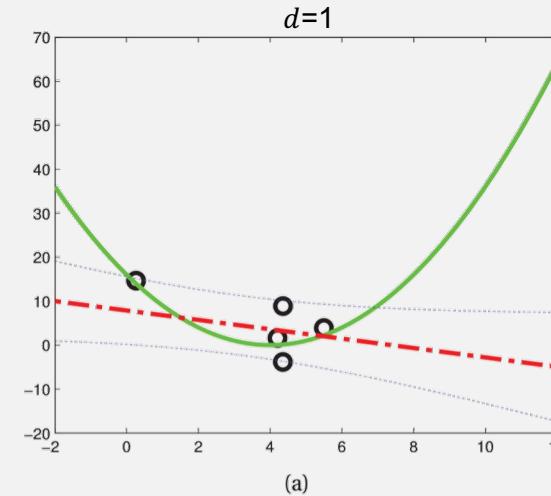
Bayesian Occam's razor

A concrete example

We plot polynomials of degrees 1, 2 and 3 fit to N=5 data points using (empirical) Bayes.

- True function
- - - Prediction
- $\pm\sigma$ around the mean

There is not enough data to justify a complex model, so the best model is $d = 1$.



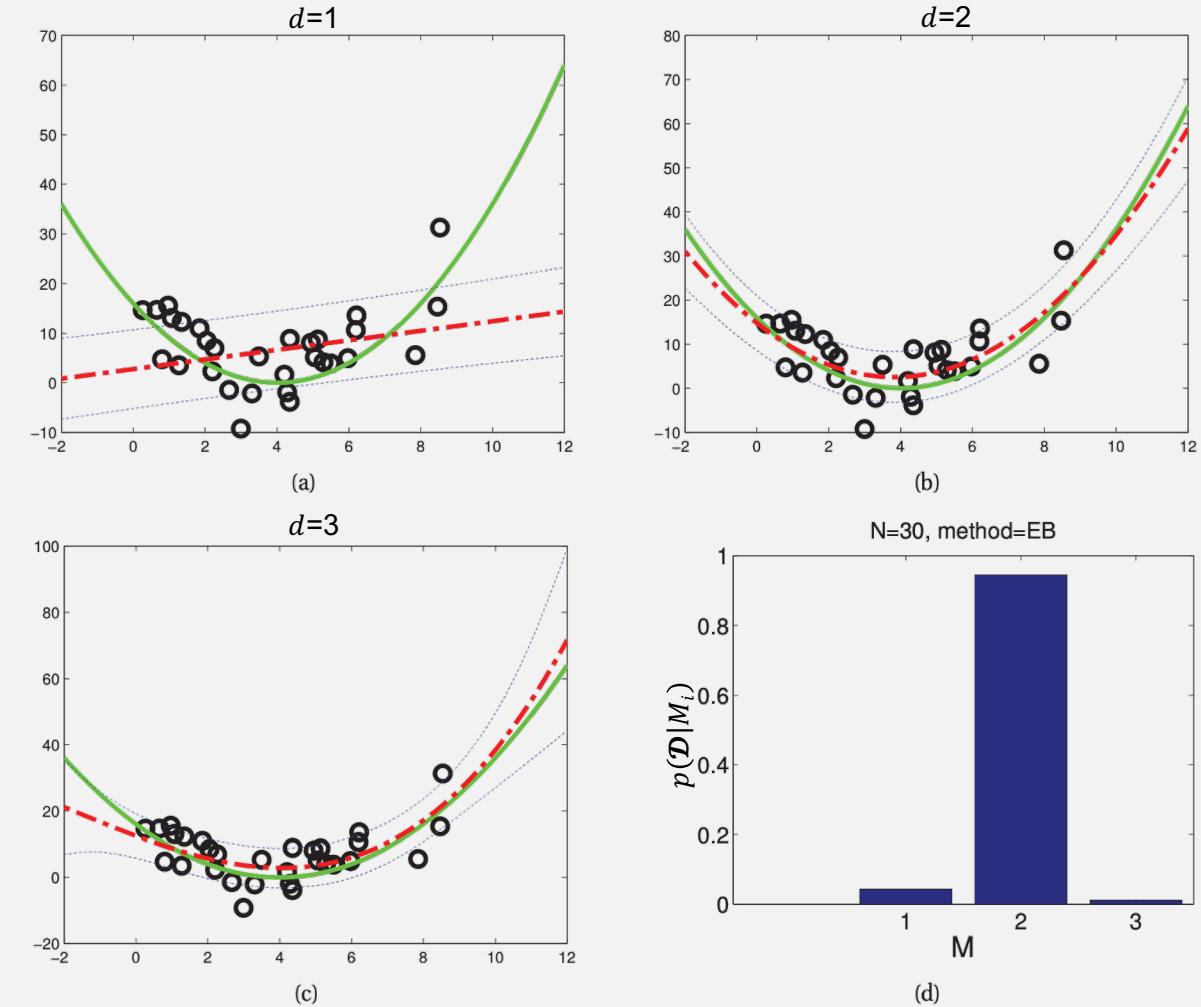
Bayesian Occam's razor

A concrete example

We plot polynomials of degrees 1, 2 and 3 fit to N=30 data points using (empirical) Bayes.

- True function
- - - Prediction
- $\pm\sigma$ around the mean

When more data is available, $d = 2$ is the right model





Bayesian Inference

Prior Distributions



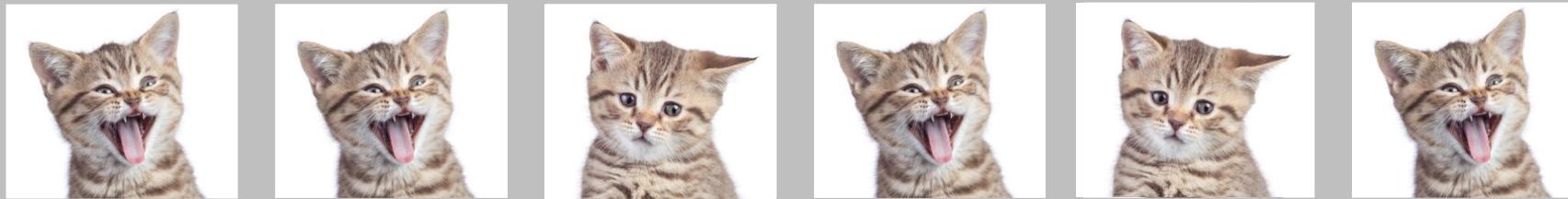




p: Prob. purring

1-p: Prob. grumpy

What is the best guess
for the probability p?



How can I update my
belief on p ?

Prior distributions

The importance of priors in Bayesian Inference

$p(\mathcal{D}|\theta)$: likelihood of θ

$p(\theta)$: prior probability of θ

$p(\theta|\mathcal{D})$: posterior of θ , given \mathcal{D}

$p(\mathcal{D})$: marginal probability of \mathcal{D}

$p(y|x, \mathcal{D})$: predictive distribution

Bayes' rule:

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta) p(\theta)}{P(\mathcal{D})}$$

$p(\mathcal{D}|\theta)$: likelihood of θ

$p(\theta)$: prior probability of θ

$p(\theta|\mathcal{D})$: posterior of θ , given \mathcal{D}

$p(\mathcal{D})$: marginal probability of \mathcal{D}

$p(y|x, \mathcal{D})$: predictive distribution

Bayes' rule:

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta) p(\theta)}{P(\mathcal{D})}$$

A **prior probability distribution** of an uncertain quantity is the probability distribution that would express one's belief, before some evidence is taken into account.

→ For example, a prior could represent the distribution of votes coming from an opinion poll, prior to the election.

A **subjective prior** expresses the modeler's subjective belief.

- We formulate our (subjective) assumptions about modeling the data in terms of priors
- We have to work hard to understand the system under study in order to formulate our assumptions

An **objective prior** constrain prior beliefs to be “uninformative” about the parameters.

- The objective Bayes view is that formulating our assumptions is too difficult, especially in complex models

If we don't have strong beliefs about what θ should be, it is common to use an “uninformative” priors → **“Let the data speak for itself!”**

An **informative prior** expresses a specific information about a variable.

- For example, a reasonable informative prior about the temperature at noon tomorrow could be given by a normal distribution with expected value equal to today's noon temperature and variance equal to the daily variance of the temperature.

An **uninformative prior** is designed to express vague or general information about a variable.

- For example, when tossing a coin, we assign the probability of 0.5 to both heads and tails.

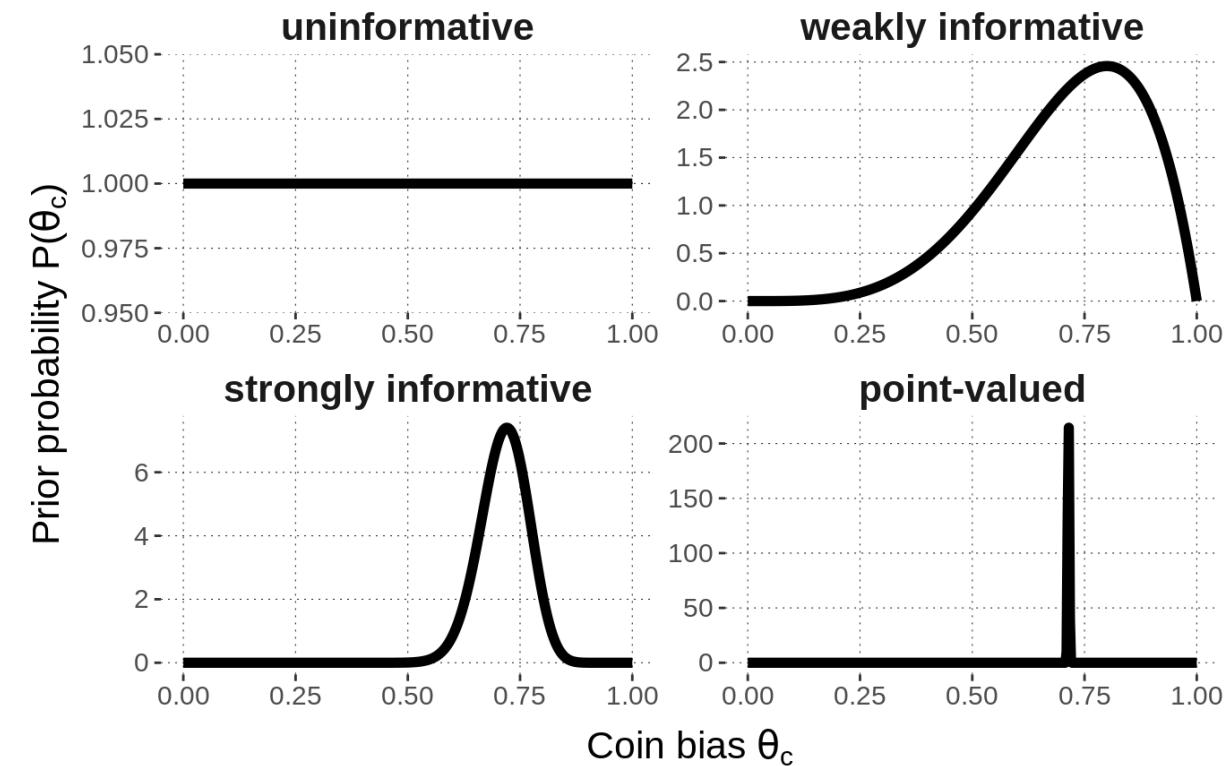


Image from: <https://michael-franke.github.io/intro-data-analysis/Chap-03-03-models-parameters-priors.html>

A key requirement for computing the posterior $p(\theta|\mathcal{D})$ is the specification of a prior $p(\theta|\alpha)$, where alpha denotes a set of hyperparameters.

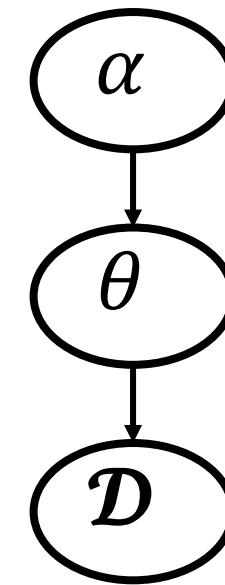
We have multiple levels of priors:

$$\alpha \rightarrow \theta \rightarrow \mathcal{D}$$

$$p(\theta) = \int p(\theta|\alpha) p(\alpha) d\alpha$$

$$p(\mathcal{D}) = \int p(\theta) p(\mathcal{D}|\theta) d\theta$$

The multi-level model



Consider the problem of predicting the cancer mortality rates in various cities. We measure the number of people N_i in various cities, as well as the number of people who died of cancer x_i in those cities. We assume that the mortality follows:

$$x_i \sim Bin(N_i, \theta_i).$$

A reasonable approach to estimate θ_i is that of assuming that they are drawn from the same distribution $\theta_i \sim Beta(a, b)$, where $\alpha = (a, b)$ are hyper-parameters in our model.

Then, the full joint distribution is written as

$$p(\mathcal{D}, \theta, \alpha | N) = p(\alpha) \prod_{i=1}^N Bin(x_i | N_i, \theta_i) Beta(\theta_i | \alpha).$$

Note: It is crucial to infer α from the data itself.

Example taken from: Machine Learning: A Probabilistic Perspective, Ch. 5.51

In hierarchical models, we need to **compute the posterior on multiple levels of variables**. For example,

$$p(\alpha, \theta | \mathcal{D}) \propto p(\mathcal{D} | \theta) p(\theta | \alpha) p(\alpha).$$

In some case, we can simplify the problems by **marginalizing over θ** . Then, we just need to compute:

$$p(\alpha | D).$$

As a computational “shortcut”, we can approximate the posterior on the hyper-parameters α with a point estimate. Since α is generally of a much smaller dimensionality than θ , it is less prone to overfitting and we safely assume a uniform prior on α .

$$\hat{\alpha} = \operatorname{argmax}_{\alpha} p(\mathcal{D}, \alpha) = \operatorname{argmax}_{\alpha} \int p(\mathcal{D}, \theta) p(\theta | \alpha) d\theta$$

A prior $p(\theta)$ is a **conjugate prior** for a particular likelihood $p(y|\theta)$ if the resulting posterior $p(\theta|y)$ has the same algebraic form.

Conjugate priors are widely used because they provide advantages:

- they usually allow us to derive a closed-form expression for the posterior distribution;
- they are easy to interpret,

Note: Conjugate priors simplify the computation, but are often not flexible enough to encode our prior knowledge → We can also use mixture of conjugate priors.

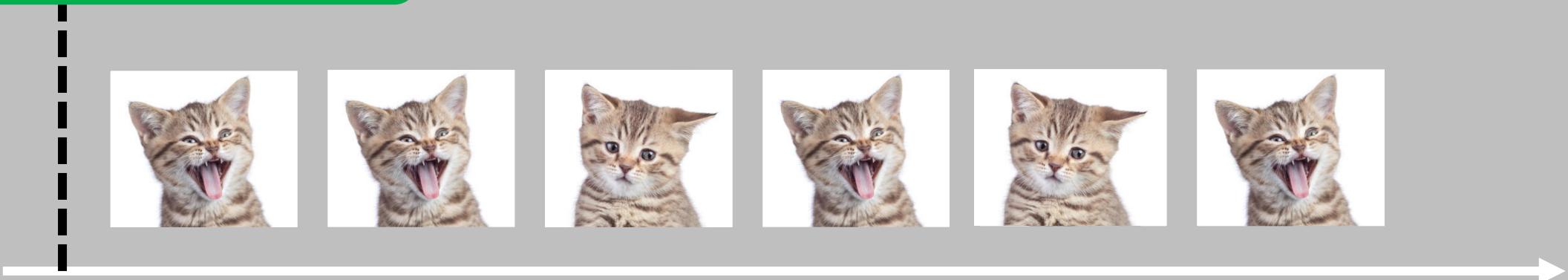
Likelihood (Binomial): $p(\mathcal{D}|\theta) = Bin(\theta) = \binom{n}{x} \theta^x (1-\theta)^{n-x}$

Prior (Beta): $p(\theta) = Beta(\alpha, \beta) = \frac{1}{B(\alpha, \beta)} \theta^{\alpha-1} (1-\theta)^{\beta-1}$

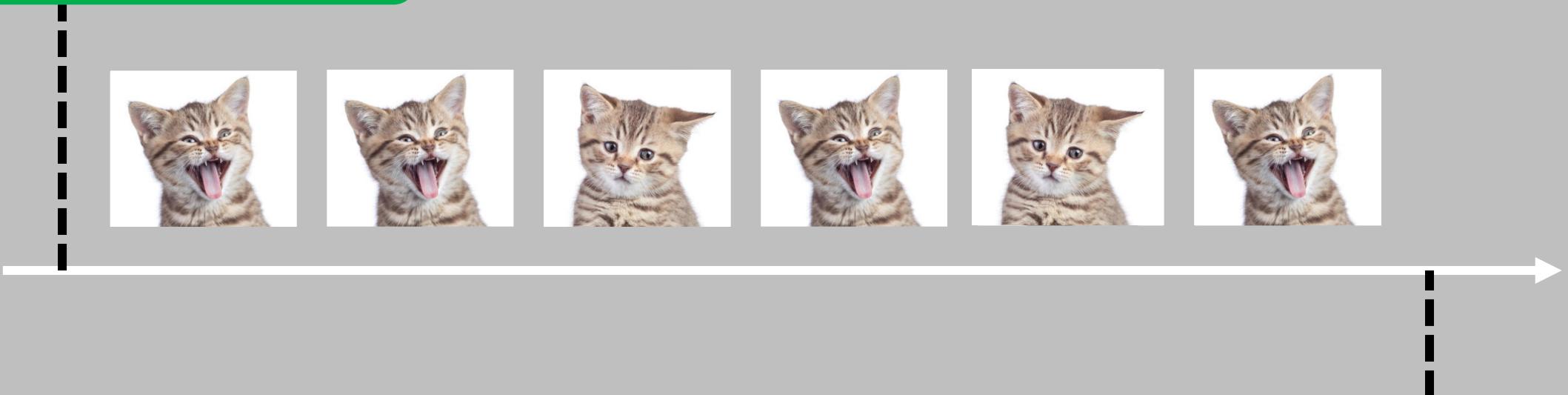
We plug them into the Bayes' formula to derive the posterior distribution:

$$\begin{aligned} p(\theta|\mathcal{D}) &= \frac{p(\mathcal{D}|\theta) p(\theta)}{\int p(\mathcal{D}|\theta) p(\theta) d\theta} \\ &= \frac{\binom{n}{x} \theta^x (1-\theta)^{n-x} \frac{1}{B(\alpha, \beta)} \theta^{\alpha-1} (1-\theta)^{\beta-1}}{\int \binom{n}{x} \theta^x (1-\theta)^{n-x} \frac{1}{B(\alpha, \beta)} \theta^{\alpha-1} (1-\theta)^{\beta-1} d\theta} \\ &= \frac{\frac{n! C_x}{B(\alpha, \beta)} \theta^{x+\alpha-1} (1-\theta)^{n-x+\beta-1}}{\frac{n! C_x}{B(\alpha, \beta)} \int \theta^{x+\alpha-1} (1-\theta)^{n-x+\beta-1} d\theta} = Beta(x + \alpha, n - x + \beta) \end{aligned}$$

Prior: Beta(2, 2)



Prior: Beta(2, 2)



Posterior: Beta($2+2, 4+2$) = Beta(4, 6)

Prior: Beta(2, 2)



Posterior: Beta($2+2, 4+2$) = Beta(4, 6)



Bayesian Inference

Linear Regression (Bayesian treatment)



Given the observed data $\mathcal{D} = \{x^{(n)}, y^{(n)}\}$, we assume to know the noise variance σ^2 .

We would like to compute the posterior over the parameters, i.e,

$$p(w|\mathcal{D}, \sigma^2).$$

(We assume throughout a Gaussian likelihood model).

In linear regression **the likelihood is given by:**

$$\begin{aligned} p(y|X, w, \mu, \sigma^2) &= \mathcal{N}(y|\mu + Xw, \sigma^2 I_N) \\ &\propto \exp\left(-\frac{1}{2\sigma^2} (y - \mu - Xw)^T (y - \mu - Xw)\right) \end{aligned}$$

where μ is an offset term.

The conjugate prior of a Gaussian likelihood is also Gaussian*, which we will denote by

$$p(w) = \mathcal{N}(w|w_0, V_0).$$

Using the Bayes rule for Gaussian*, the posterior is given by

$$p(w|X, y, \sigma^2) \propto \mathcal{N}(w|w_0, V_0) \mathcal{N}(y|Xw, \sigma^2 I_N) = \mathcal{N}(w|w_N, V_N)$$

where

$$w_N = V_N V_0^{-1} w_0 + \frac{1}{\sigma^2} V_N X^T y$$

$$V_N = \sigma^2 (\sigma^2 V_0^{-1} + X^T X)^{-1}$$

* See: Murphy K., „Machine Learning: A Probabilistic Perspective“ (2012)

The posterior predictive distribution at a test point x is given by *

$$\begin{aligned} p(y|x, \mathcal{D}, \sigma^2) &= \int \mathcal{N}(y|x^T w, \sigma^2) \mathcal{N}(w|w_N, V_N) dw \\ &= \mathcal{N}(y|w_N^T x, \sigma_N^2(x)) \end{aligned}$$

where $\sigma_N^2(x) = \sigma^2 + x^T V_N x$.

The variance in this prediction depends on the variance of the observation noise, σ^2 , and the variance in the parameters, V_N .



Bayesian Inference

Recap



- Bayesian modelling
 - Prior
 - Posterior
 - Likelihood
 - Priors
 - Informative vs Uninformative
 - Conjugate priors
 - Linear regression with Bayesian treatment
- Bayesian modelling requires integration over parameters
 - For complex models it could be not tractable! (We cannot compute the integral analytically)



Machine Learning for Time Series

(MLTS or MLTS-Deluxe Lectures)

Dr. Dario Zanca

Machine Learning and Data Analytics (MaD) Lab
Friedrich-Alexander-Universität Erlangen-Nürnberg
25.10.2022

- Time series fundamentals and definitions (2 lectures)
- Bayesian Inference (1 lecture)
- Gaussian processes (2 lectures) ←
- State space models (2 lectures)
- Autoregressive models (1 lecture)
- Data mining on time series (1 lecture)
- Deep learning on time series (4 lectures)
- Domain adaptation (1 lecture)

In this lecture...

1. Prior on functions
2. Gaussian processes
3. Gaussian process regression



Gaussian Process Regression

Prior on functions

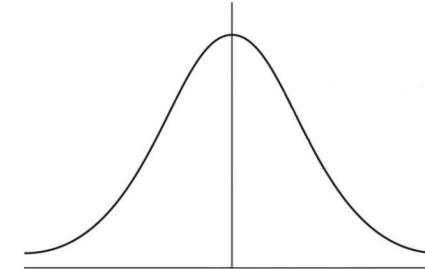


Concepts review: Gaussian Distribution

Univariate vs. Multivariate

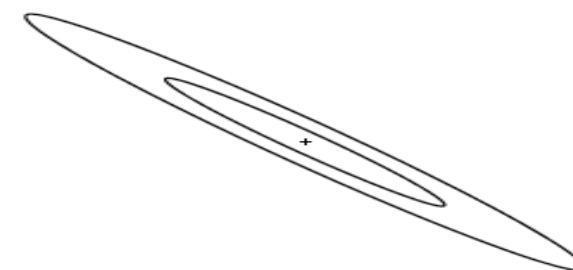
The **univariate Gaussian distribution** is given by

$$\mathcal{N}(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(x-\mu)^2}$$



The **multivariate Gaussian distribution** is given
by

$$\mathcal{N}(x|\mu, \Sigma) = (2\pi)^{-D/2} |\Sigma|^{-1/2} e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)}$$



Concept review: Conditional and Marginal of a Gaussian

If x and y are jointly Gaussian

$$p(x, y) = p\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix} a \\ b \end{bmatrix}, \begin{bmatrix} A & B \\ B^T & C \end{bmatrix}\right)$$

we get the marginal distribution of x by

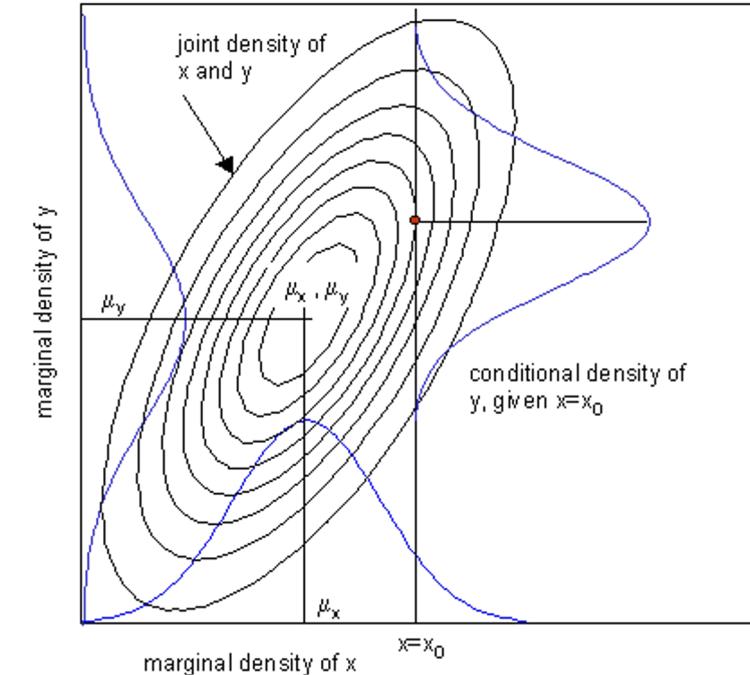
$$p(x, y) = \mathcal{N}\left(\begin{bmatrix} a \\ b \end{bmatrix}, \begin{bmatrix} A & B \\ B^T & C \end{bmatrix}\right) \Rightarrow p(x) = \mathcal{N}(a, A)$$

and the conditional distribution of x given y by

$$p(x, y) = \mathcal{N}\left(\begin{bmatrix} a \\ b \end{bmatrix}, \begin{bmatrix} A & B \\ B^T & C \end{bmatrix}\right) \Rightarrow p(x|y) = \mathcal{N}(a + BC^{-1}(y - b), A - BC^{-1}B^T)$$

where x and y can be scalars or vectors.

Both the conditional $p(x|y)$ and the marginal $p(x)$ of a joint Gaussian $p(x, y)$ are Gaussian.



In **supervised learning**, we:

- observe some inputs x_i and some outputs y_i
- Assume that $y_i = f(x_i)$
 - for some unknown function f
 - Possibly subject to noise

The optimal approach is to:

- infer a distribution over functions given the data, $p(f | X, y)$
- Then use it for prediction

$$p(y_*|x_*, X, y) = \int p(y_*|f, x_*)p(f|X, y) df$$

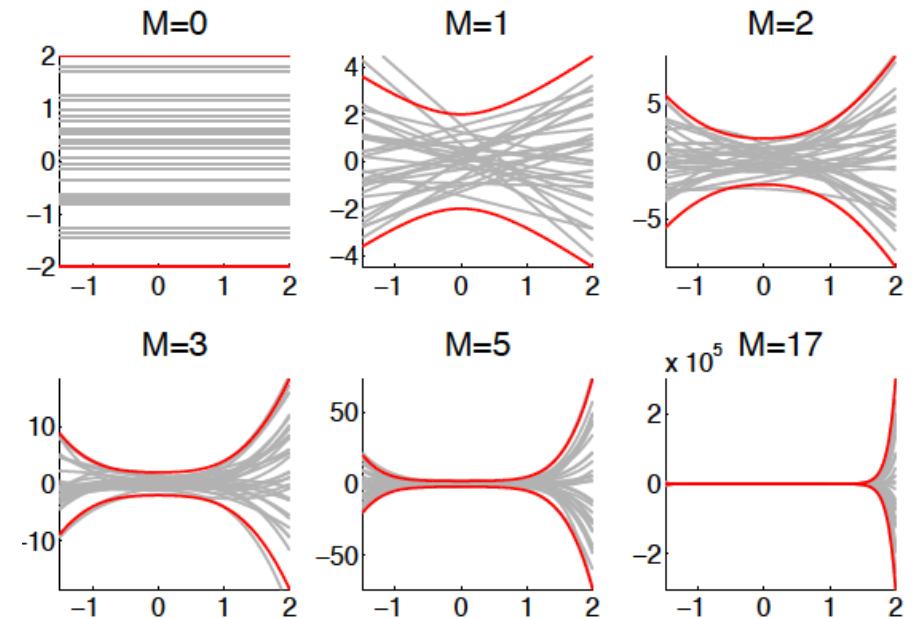
A model M is the result of the choice of:

- A **model structure**
- The **model's parameters**

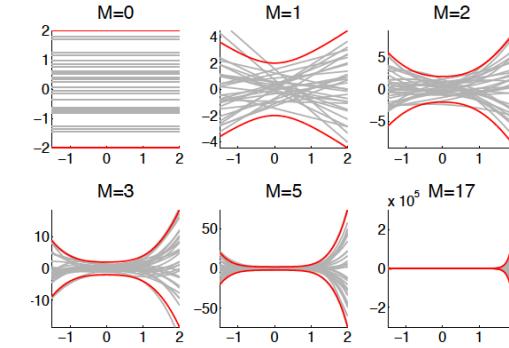
In the example:

$$f_w(x) = \sum_{m=0}^M \omega_m \Phi_m(x), \text{ with } \Phi_m(x) = x^m$$

We have defined a prior distribution over functions
but **in an indirect way**

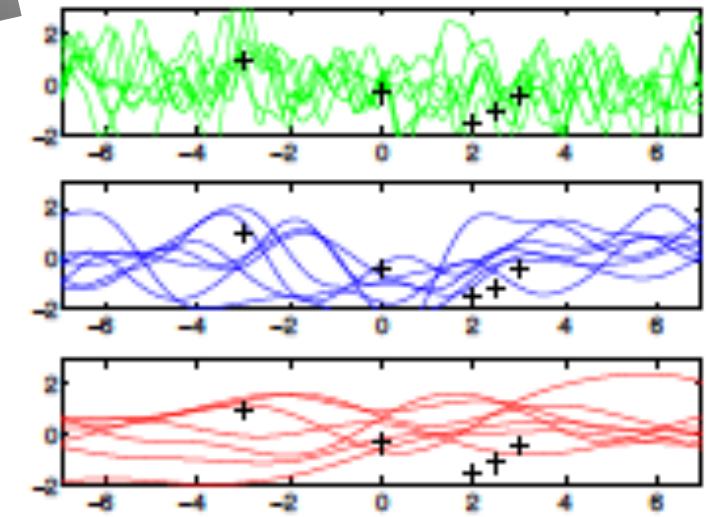
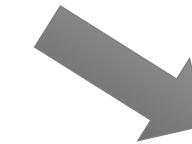
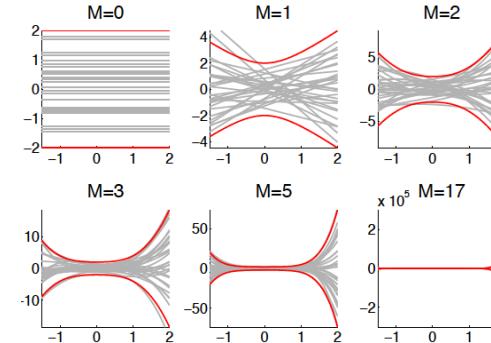


Models with priors on the weights *indirectly* specify priors over functions.



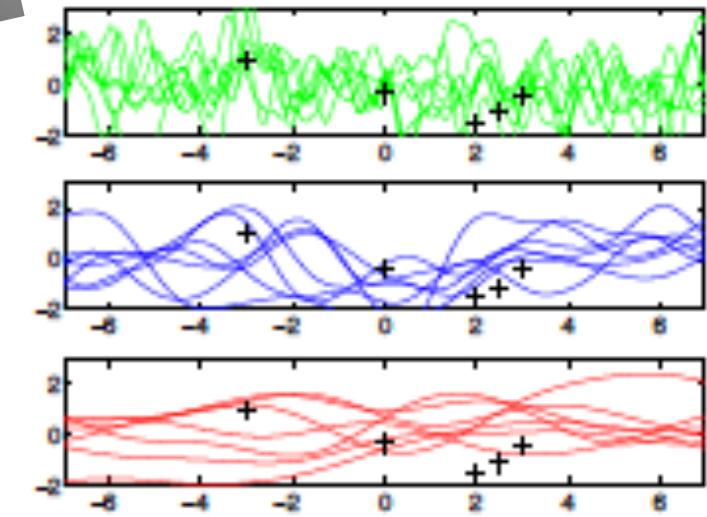
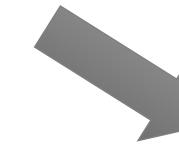
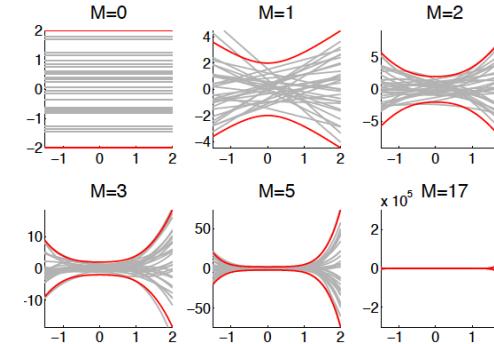
Models with priors on the weights *indirectly* specify priors over functions.

- **What about specifying priors on functions directly?**
- **What does a probability density over functions even look like?**



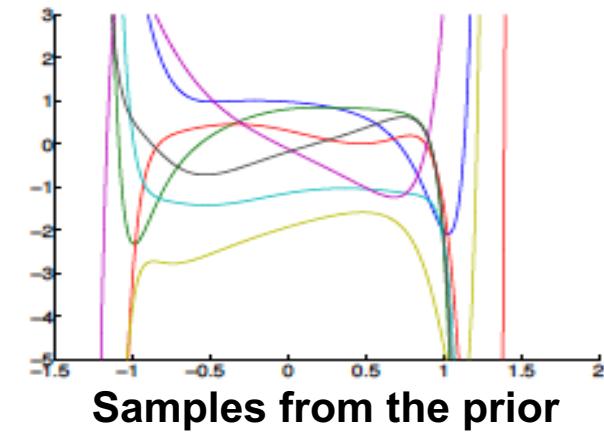
Models with priors on the weights *indirectly* specify priors over functions.

- **What about specifying priors on functions directly?**
- **What does a probability density over functions even look like?**



The Bayes rule can be written as:

$$p(f|y) = \frac{p(y|f)p(f)}{p(y)}$$

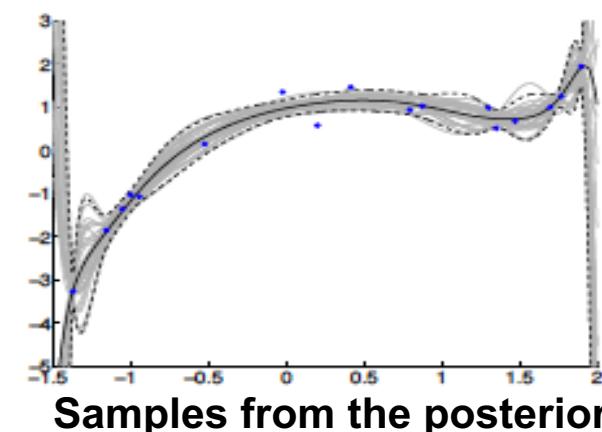
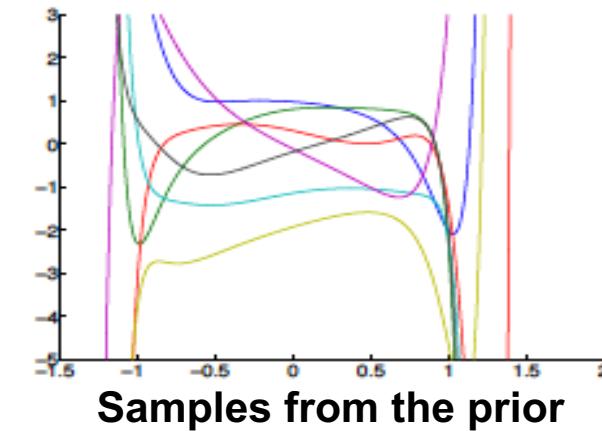


The Bayes rule can be written as:

$$p(f|y) = \frac{p(y|f)p(f)}{p(y)}$$

We keep the functions which are “closer” to the data

→ Notion of **closeness** is given by the likelihood $p(y|f)$



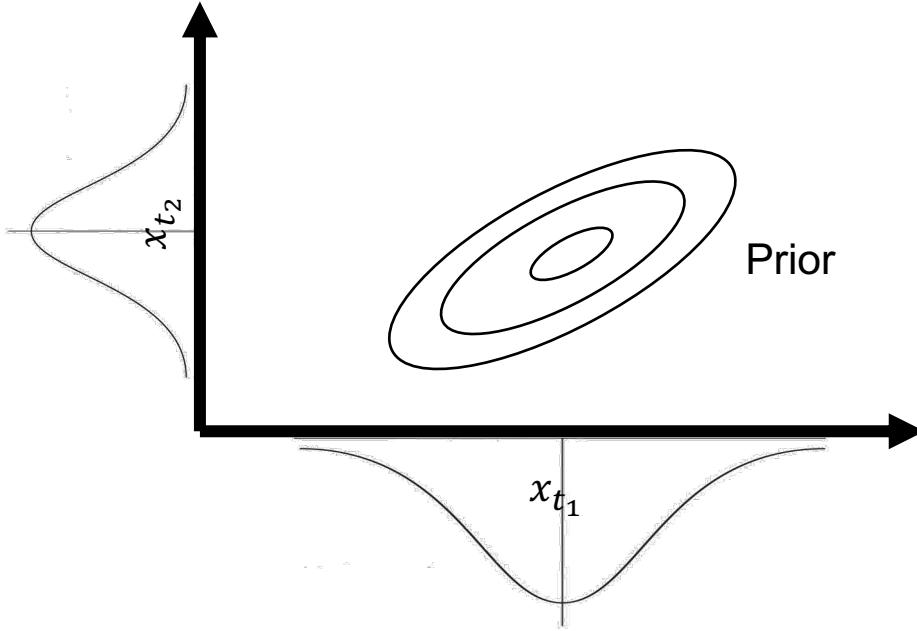


Gaussian Process Regression

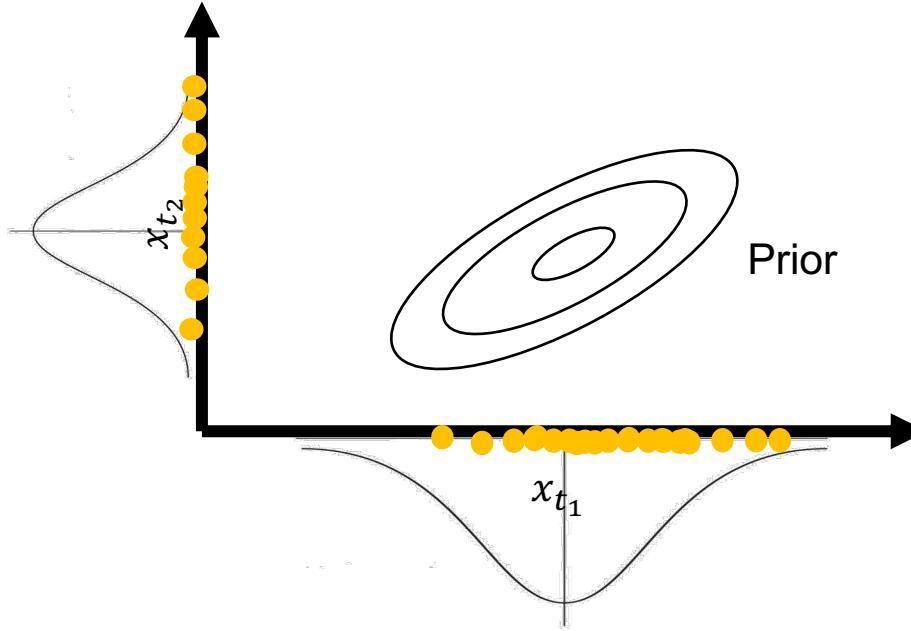
Gaussian Processes (GP)



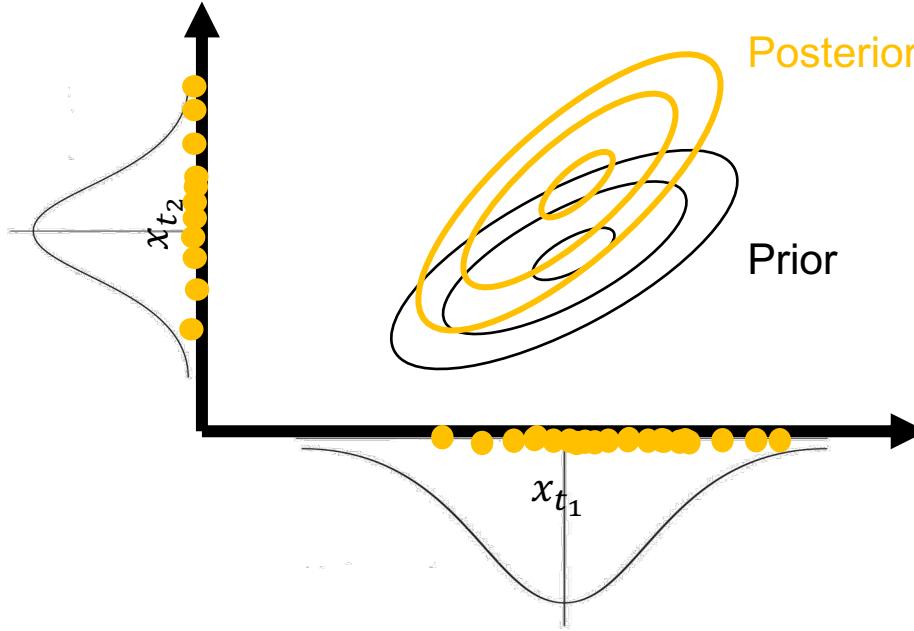
Towards Gaussian Processes

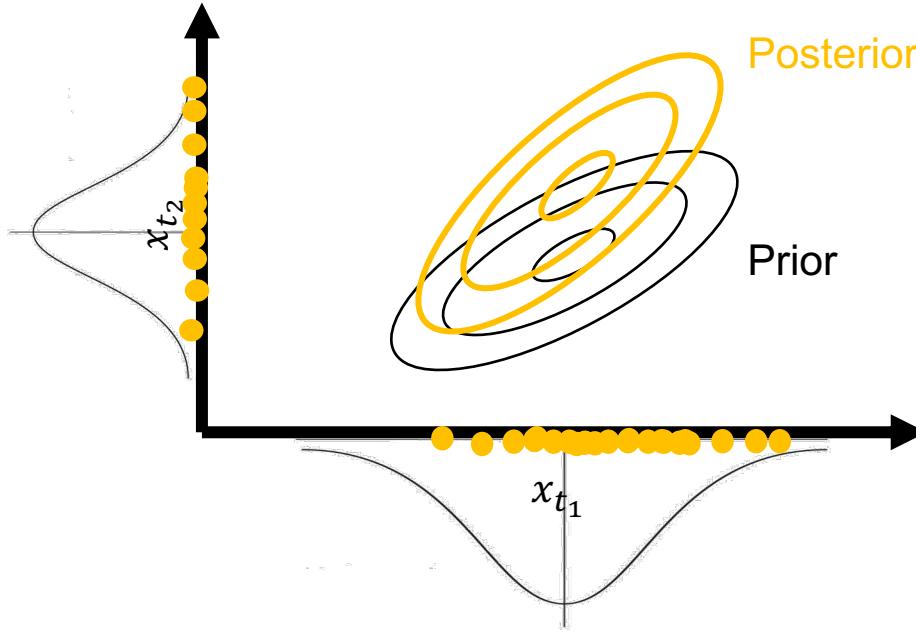


Towards Gaussian Processes

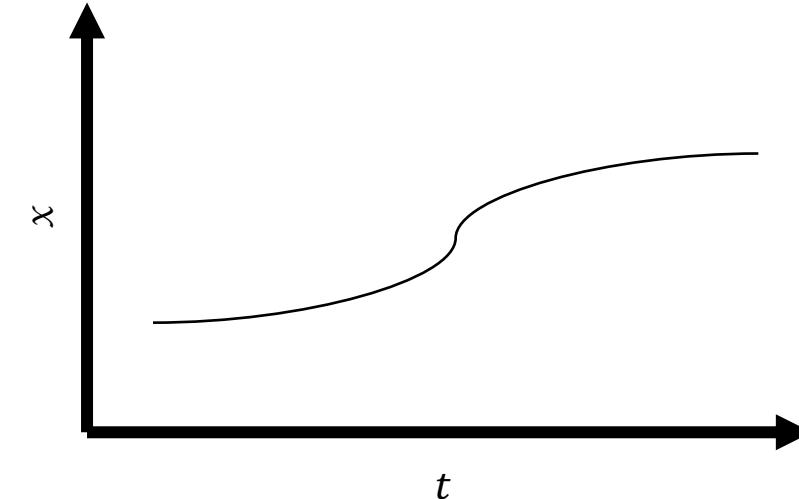
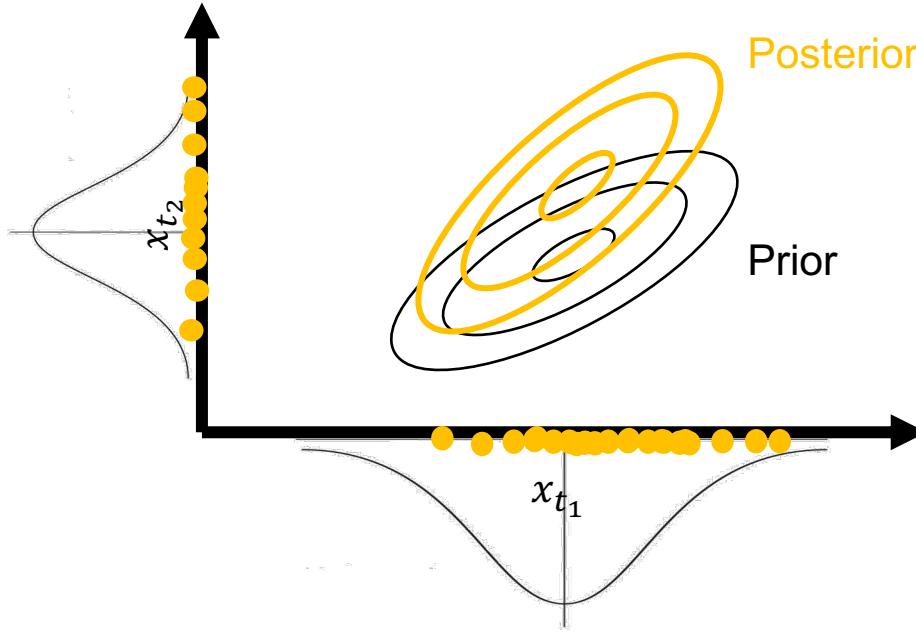


Towards Gaussian Processes

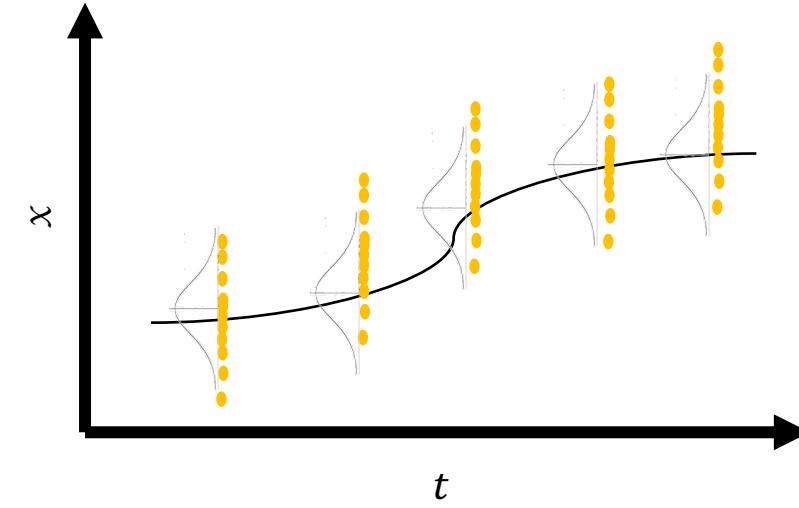
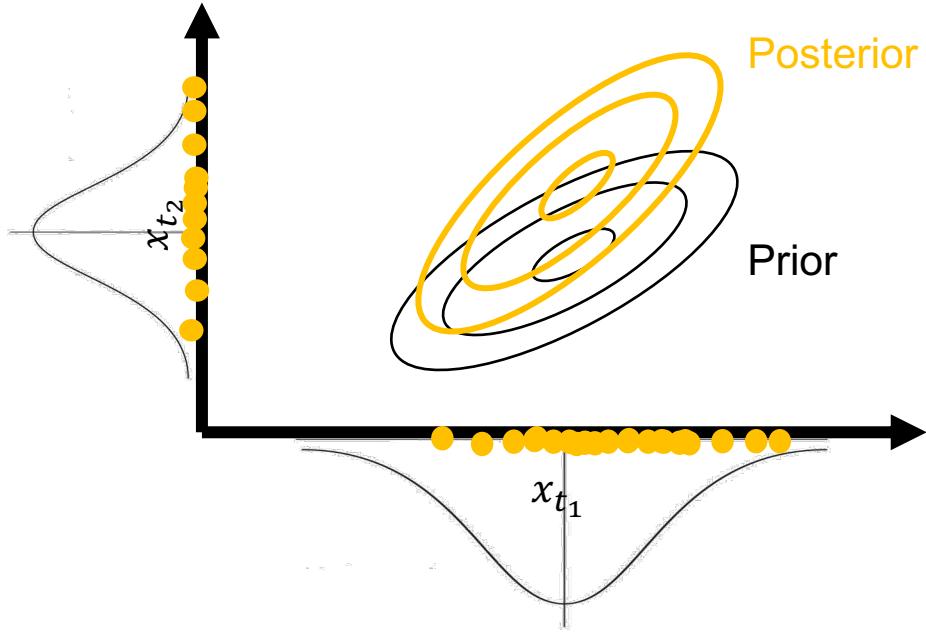




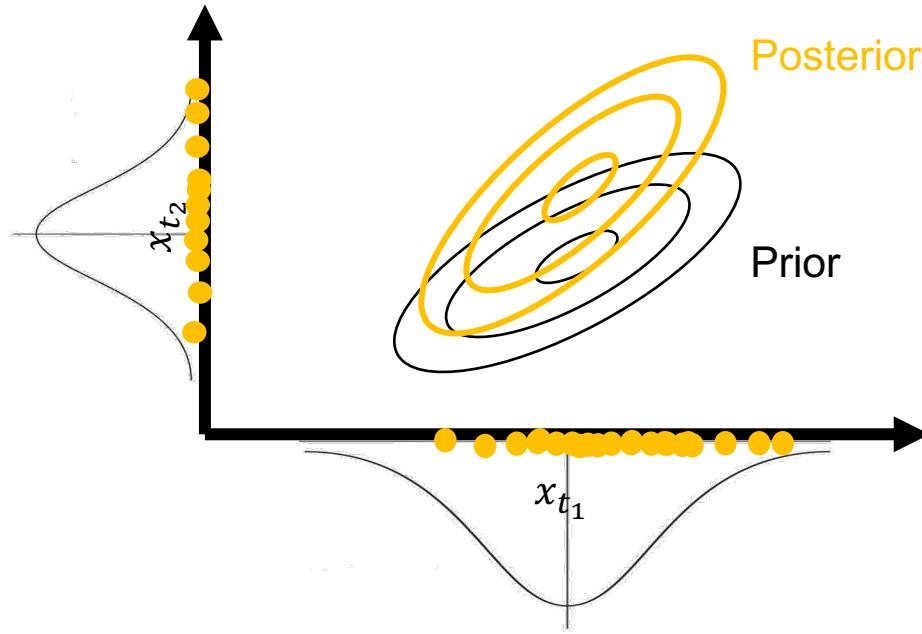
For **multivariate Gaussian distributions** we look at groups of real-valued variables.



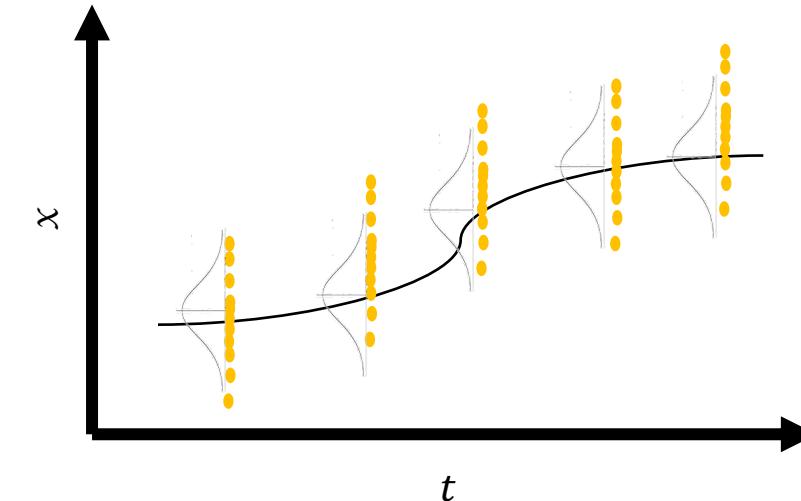
For **multivariate Gaussian distributions** we look at groups of real-valued variables.



For **multivariate Gaussian distributions** we look at groups of real-valued variables.



For **multivariate Gaussian distributions** we look at groups of real-valued variables.



For **Gaussian processes** we look at very many random variables with Gaussian distribution.
→ GP are functions of (potentially infinite) number of real-valued variables.

Definition:

A function f is a Gaussian process if $f(t) = [f(t_1), \dots, f(t_N)]^T$ has multivariate distribution for each $t = [t_1, \dots, t_N]^T$.

For any subset of t : $f(t) \sim N(\mu(t), \Sigma(t, t'))$

Notice: here we use t for time, but in general we can have a $x \in \mathbb{R}^d$.

The **mean function** is defined as

$$\mu: \mathbb{R} \rightarrow \mathbb{R} \quad (\text{or}, \mathbb{R}^d \rightarrow \mathbb{R})$$

- Often, we subtract the mean from the data to have $\mu(t) = 0, \forall t$

The **covariance function** is defined as $\Sigma: \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}^d$; positive semidefinite matrix.

- Often for GP we refer to a **kernel function** $k(t, t')$.

Notice: here we use t for time, but in general we can have a $x \in \mathbb{R}^d$.

The **mean function** is defined as

$$\mu: \mathbb{R} \rightarrow \mathbb{R} \quad (\text{or}, \mathbb{R}^d \rightarrow \mathbb{R})$$

- Often, we subtract the mean from the data to have $\mu(t) = 0, \forall t$

The **covariance function** is defined as $\Sigma: \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}^d$; positive semidefinite matrix.

- Often for GP we refer to a **kernel function** $k(t, t')$.

We can, then, rewrite the Gaussian process as:

- $f(t) \sim \mathcal{N}(\mu(t), k(t, t'))$ or $f(t) \sim \mathcal{N}(0, k(t, t'))$

Notice: here we use t for time, but in general we can have a $x \in \mathbb{R}^d$.

The **mean function** is defined as

$$\mu: \mathbb{R} \rightarrow \mathbb{R} \quad (\text{or}, \mathbb{R}^d \rightarrow \mathbb{R})$$

- Often, we subtract the mean from the data to have $\mu(t) = 0, \forall t$

The **covariance function** is defined as $\Sigma: \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}^d$; positive semidefinite matrix.

- Often for GP we refer to a **kernel function** $k(t, t')$

A GP is defined by its mean and kernel function, so we can write:

$$f \sim GP(\mu, k)$$

We can, then, rewrite the Gaussian process as:

- $f(t) \sim \mathcal{N}(\mu(t), k(t, t'))$ or $f(t) \sim \mathcal{N}(0, k(t, t'))$

Gaussian process

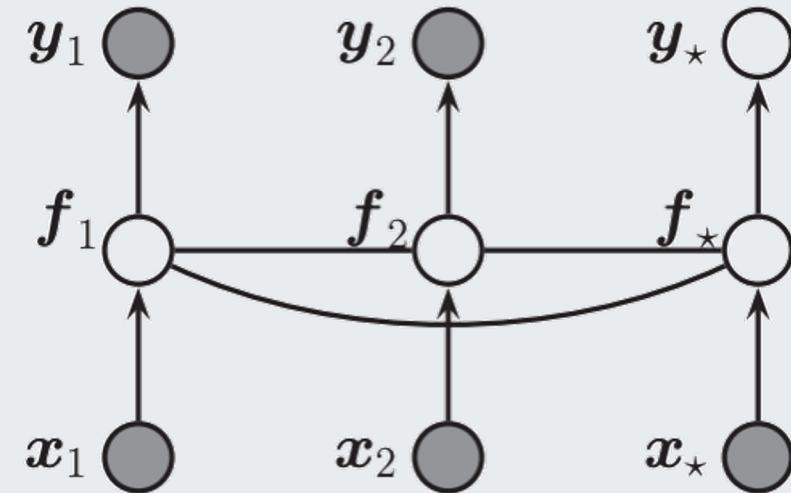
Graphical illustration

Then, a Gaussian Process assumes that the distribution over the function's on a finite (and arbitrary) set of points,

$$p(f(x_1), \dots, f(x_N)),$$

is jointly Gaussian, with mean $\mu(x)$ and covariance $\Sigma(x)$, where the covariance is given by $\Sigma_{ij} = \kappa(x_i, x_j)$ and κ being a positive definite kernel function.

Key idea: if x_i and x_j are similar w.r.t. the kernel, their output through f will also be similar.



**Gaussian process
(graphical illustration)**

$$p(y, f | x) = \mathcal{N}(0, \kappa(x)) \prod_i p(x_i, f_i)$$

Example of Gaussian process

Random lines

Let's $t = \mathbb{R}$, be the entire real line. We define:

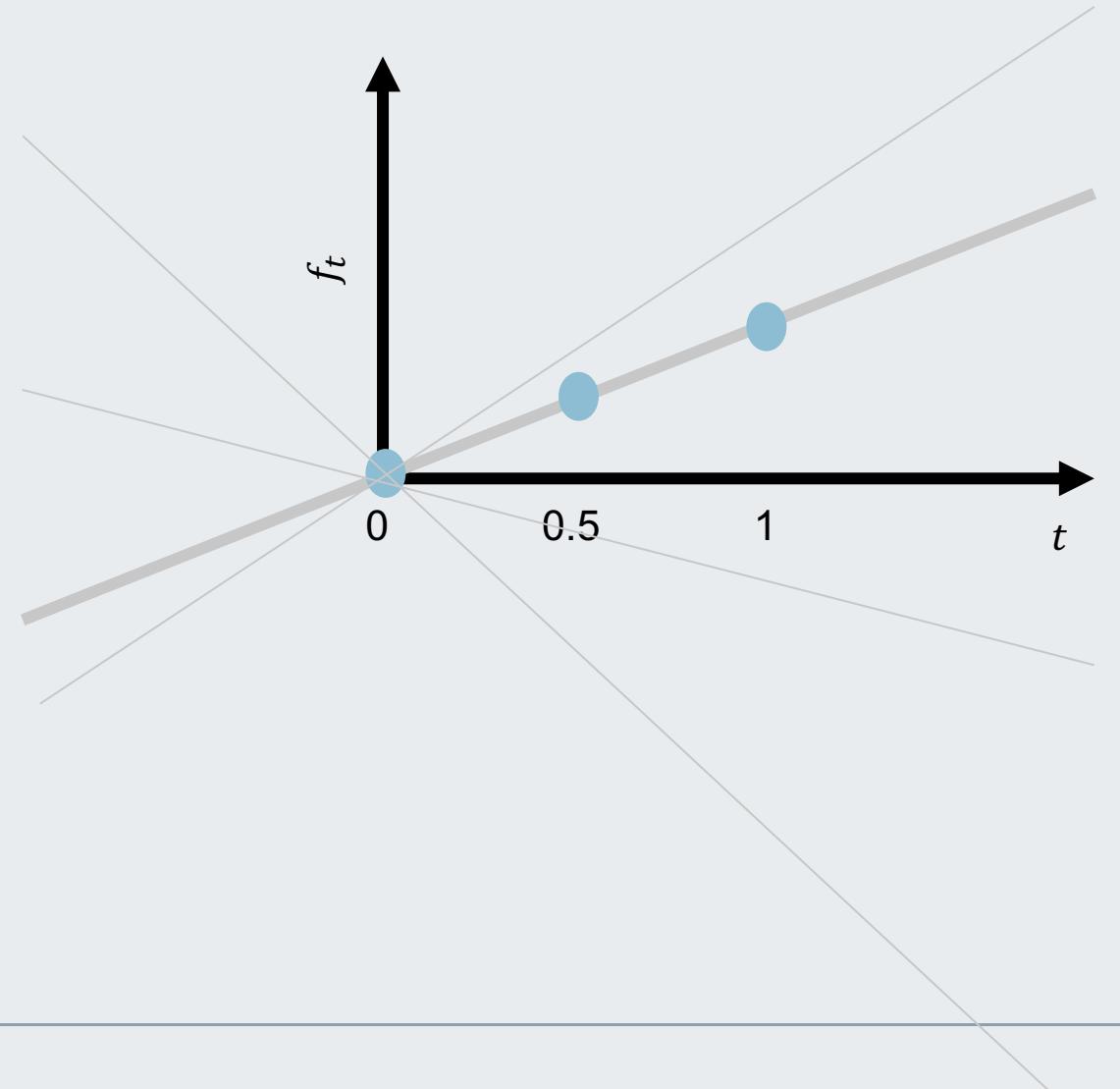
$$f_t = t \cdot w$$

with $w \sim \mathcal{N}(0,1)$, $w \in \mathbb{R}$.

We verify that f is a GP:

$$\begin{aligned}[f_{t_1}, \dots, f_{t_N}]^T &= \\ &= [wt_1, \dots, wt_N]^T = \\ &= w[t_1, \dots, t_N]^T\end{aligned}$$

→ Since $w \sim \mathcal{N}(0,1)$ is (multivariate) Gaussian, the result is also (multivariate) Gaussian.





Gaussian Process Regression

Gaussian Processes (GP) regression



Suppose:

$$\begin{bmatrix} f \\ y \end{bmatrix} \sim \mathcal{N} \left(\begin{bmatrix} \mu_f \\ \mu_y \end{bmatrix}, \begin{bmatrix} \Sigma_{ff} & \Sigma_{fy} \\ \Sigma_{fy}^T & \Sigma_{yy} \end{bmatrix} \right)$$

Then,

$$p(f|y) = \mathcal{N}(\mu_{f|y}, \Sigma_{f|y})$$

where:

- $\mu_{f|y} = \mu_f + \Sigma_{fy}\Sigma_{yy}^{-1}(y - \mu_y)$
- $\Sigma_{f|y} = \Sigma_{ff} - \Sigma_{fy}\Sigma_{yy}^{-1}\Sigma_{fy}^T$

Suppose:

$$\begin{bmatrix} f \\ y \end{bmatrix} \sim \mathcal{N} \left(\begin{bmatrix} \mu_f \\ \mu_y \end{bmatrix}, \begin{bmatrix} \Sigma_{ff} & \Sigma_{fy} \\ \Sigma_{fy}^T & \Sigma_{yy} \end{bmatrix} \right)$$

Then,

$$p(f|y) = \mathcal{N}(\mu_{f|y}, \Sigma_{f|y})$$

where:

- $\mu_{f|y} = \mu_f + \Sigma_{fy}\Sigma_{yy}^{-1}(y - \mu_y)$
- $\Sigma_{f|y} = \Sigma_{ff} - \Sigma_{fy}\Sigma_{yy}^{-1}\Sigma_{fy}^T$

The mean “update” is linear function of the observation y

Suppose:

$$\begin{bmatrix} f \\ y \end{bmatrix} \sim \mathcal{N} \left(\begin{bmatrix} \mu_f \\ \mu_y \end{bmatrix}, \begin{bmatrix} \Sigma_{ff} & \Sigma_{fy} \\ \Sigma_{fy}^T & \Sigma_{yy} \end{bmatrix} \right)$$

Then,

$$p(f|y) = \mathcal{N}(\mu_{f|y}, \Sigma_{f|y})$$

where:

- $\mu_{f|y} = \mu_f + \Sigma_{fy}\Sigma_{yy}^{-1}(y - \mu_y)$
- $\Sigma_{f|y} = \Sigma_{ff} - \Sigma_{fy}\Sigma_{yy}^{-1}\Sigma_{fy}^T$

The mean “update” is linear function of the observation y

How much does the data explain?

Small → it can approach 0 → Uncertain $\sim \Sigma_{ff}$

Large → it can approach Σ_{ff} → zero covariance!

Suppose:

$$\begin{bmatrix} y \\ y^* \end{bmatrix} \sim \mathcal{N} \left(\begin{bmatrix} \mu_y \\ \mu_{y^*} \end{bmatrix}, \begin{bmatrix} \Sigma_{yy} & \Sigma_{y^*y} \\ \Sigma_{y^*y}^T & \Sigma_{y^*y^*} \end{bmatrix} \right)$$

Then,

$$p(y^*|y) = \mathcal{N}(\mu_{y^*|y}, \Sigma_{y^*|y})$$

where:

- $\mu_{y^*|y} = \mu_{y^*} + \Sigma_{y^*y}\Sigma_{yy}^{-1}(y - \mu_y)$
- $\Sigma_{y^*|y} = \Sigma_{y^*y^*} - \Sigma_{y^*y}\Sigma_{yy}^{-1}\Sigma_{y^*y}^T$

Suppose:

$$\begin{bmatrix} y \\ y^* \end{bmatrix} \sim \mathcal{N} \left(\begin{bmatrix} \mu_y \\ \mu_{y^*} \end{bmatrix}, \begin{bmatrix} \Sigma_{yy} & \Sigma_{y^*y} \\ \Sigma_{y^*y}^T & \Sigma_{y^*y^*} \end{bmatrix} \right)$$

Then,

$$p(y^*|y) = \mathcal{N}(\mu_{y^*|y}, \Sigma_{y^*|y})$$

where:

- $\mu_{y^*|y} = \mu_{y^*} + \Sigma_{y^*y}\Sigma_{yy}^{-1}(y - \mu_y)$

- $\Sigma_{y^*|y} = \Sigma_{y^*y^*} - \Sigma_{y^*y}\Sigma_{yy}^{-1}\Sigma_{y^*y}^T$

Predictive mean

Predictive covariance

Function to be estimated: $y(t) = t \sin(t)$

Sampling interval: $t \in [0, 10]$

Remember the conditioning:

$$p(f|y) = \mathcal{N}(\mu_{f|y}, \Sigma_{f|y})$$

where:

- $\mu_{f|y} = \mu_f + \Sigma_{fy}\Sigma_{yy}^{-1}(y - \mu_y)$
- $\Sigma_{f|y} = \Sigma_{ff} - \Sigma_{fy}\Sigma_{yy}^{-1}\Sigma_{fy}^T$

Example for GP regression

Function to be estimated: $y(t) = t \sin(t)$

Sampling interval: $t \in [0, 10]$

Remember the conditioning:

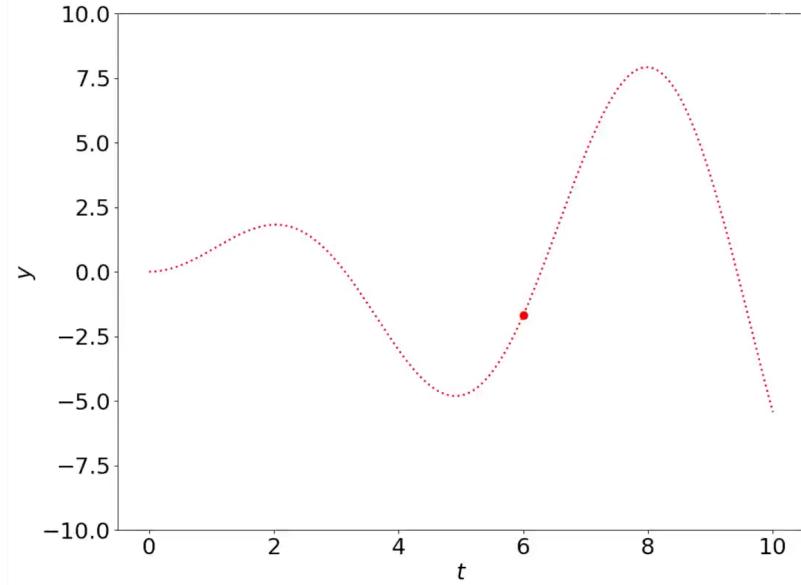
$$p(f|y) = \mathcal{N}(\mu_{f|y}, \Sigma_{f|y})$$

where:

- $\mu_{f|y} = \mu_f + \Sigma_{fy}\Sigma_{yy}^{-1}(y - \mu_y)$
- $\Sigma_{f|y} = \Sigma_{ff} - \Sigma_{fy}\Sigma_{yy}^{-1}\Sigma_{fy}^T$

Now, we can plug in particular values for y , e.g.,

$$p(f|\mathbf{y}(t=6)) = \mathcal{N}(\mu_{f|\mathbf{y}(t=6)}, \Sigma_{f|\mathbf{y}(t=6)})$$



Example for GP regression

Function to be estimated: $y(t) = t \sin(t)$

Sampling interval: $t \in [0, 10]$

Remember the conditioning:

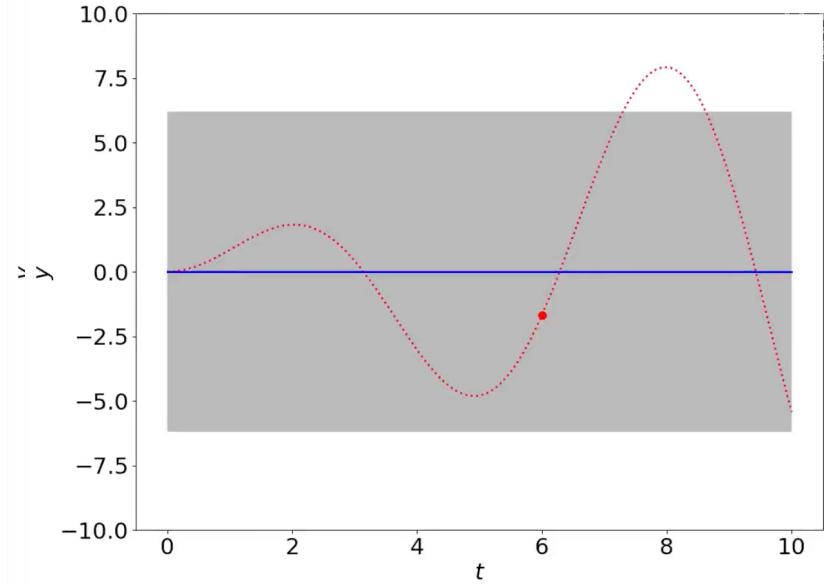
$$p(f|y) = \mathcal{N}(\mu_{f|y}, \Sigma_{f|y})$$

where:

- $\mu_{f|y} = \mu_f + \Sigma_{fy}\Sigma_{yy}^{-1}(y - \mu_y)$
- $\Sigma_{f|y} = \Sigma_{ff} - \Sigma_{fy}\Sigma_{yy}^{-1}\Sigma_{fy}^T$

Now, we can plug in particular values for y , e.g.,

$$p(f|\mathbf{y}(t=6)) = \mathcal{N}(\mu_{f|\mathbf{y}(t=6)}, \Sigma_{f|\mathbf{y}(t=6)})$$



Example for GP regression

Function to be estimated: $y(t) = t \sin(t)$

Sampling interval: $t \in [0, 10]$

Remember the conditioning:

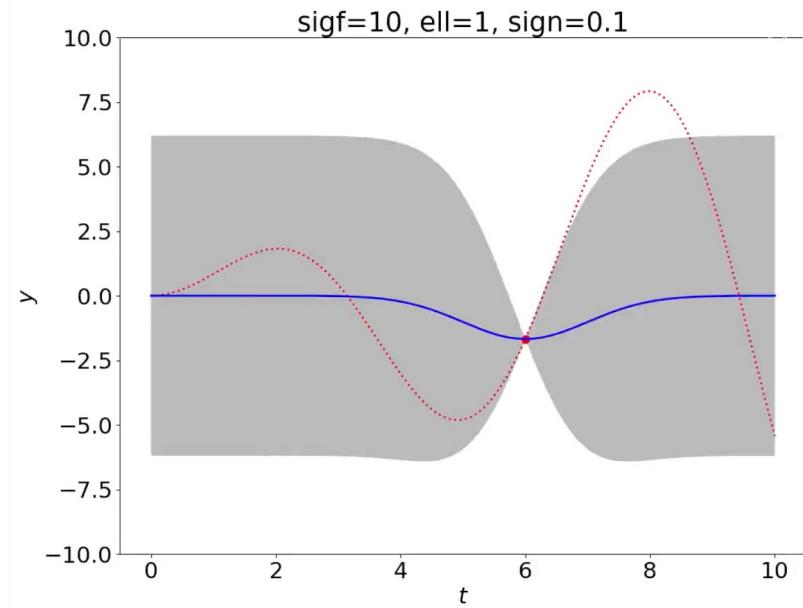
$$p(f|y) = \mathcal{N}(\mu_{f|y}, \Sigma_{f|y})$$

where:

- $\mu_{f|y} = \mu_f + \Sigma_{fy}\Sigma_{yy}^{-1}(y - \mu_y)$
- $\Sigma_{f|y} = \Sigma_{ff} - \Sigma_{fy}\Sigma_{yy}^{-1}\Sigma_{fy}^T$

Now, we can plug in particular values for y , e.g.,

$$p(f|\mathbf{y}(t=6)) = \mathcal{N}(\mu_{f|\mathbf{y}(t=6)}, \Sigma_{f|\mathbf{y}(t=6)})$$



Example for GP regression

Function to be estimated: $y(t) = t \sin(t)$

Sampling interval: $t \in [0, 10]$

Remember the conditioning:

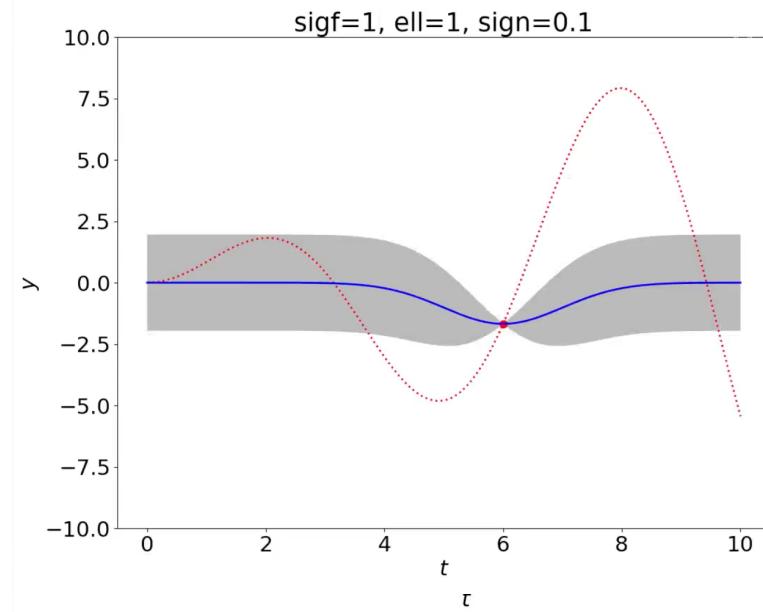
$$p(f|y) = \mathcal{N}(\mu_{f|y}, \Sigma_{f|y})$$

where:

- $\mu_{f|y} = \mu_f + \Sigma_{fy}\Sigma_{yy}^{-1}(y - \mu_y)$
- $\Sigma_{f|y} = \Sigma_{ff} - \Sigma_{fy}\Sigma_{yy}^{-1}\Sigma_{fy}^T$

Now, we can plug in particular values for y , e.g.,

$$p(f|\mathbf{y}(t=6)) = \mathcal{N}(\mu_{f|\mathbf{y}(t=6)}, \Sigma_{f|\mathbf{y}(t=6)})$$



Example for GP regression

Function to be estimated: $y(t) = t \sin(t)$

Sampling interval: $t \in [0, 10]$

Remember the conditioning:

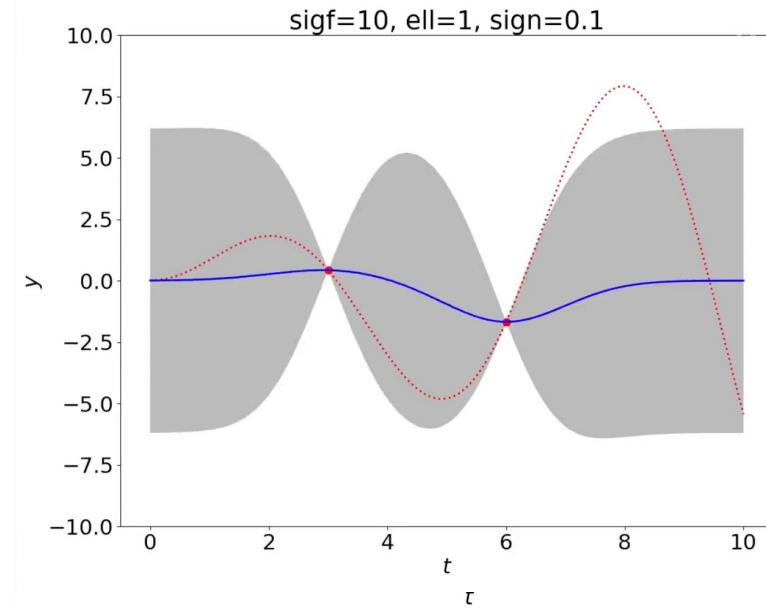
$$p(f|y) = \mathcal{N}(\mu_{f|y}, \Sigma_{f|y})$$

where:

- $\mu_{f|y} = \mu_f + \Sigma_{fy}\Sigma_{yy}^{-1}(y - \mu_y)$
- $\Sigma_{f|y} = \Sigma_{ff} - \Sigma_{fy}\Sigma_{yy}^{-1}\Sigma_{fy}^T$

Now, we can plug in particular values for y , e.g.,

$$p(f|\mathbf{y}(t=6)) = \mathcal{N}(\mu_{f|\mathbf{y}(t=6)}, \Sigma_{f|\mathbf{y}(t=6)})$$



Example for GP regression

Function to be estimated: $y(t) = t \sin(t)$

Sampling interval: $t \in [0, 10]$

Remember the conditioning:

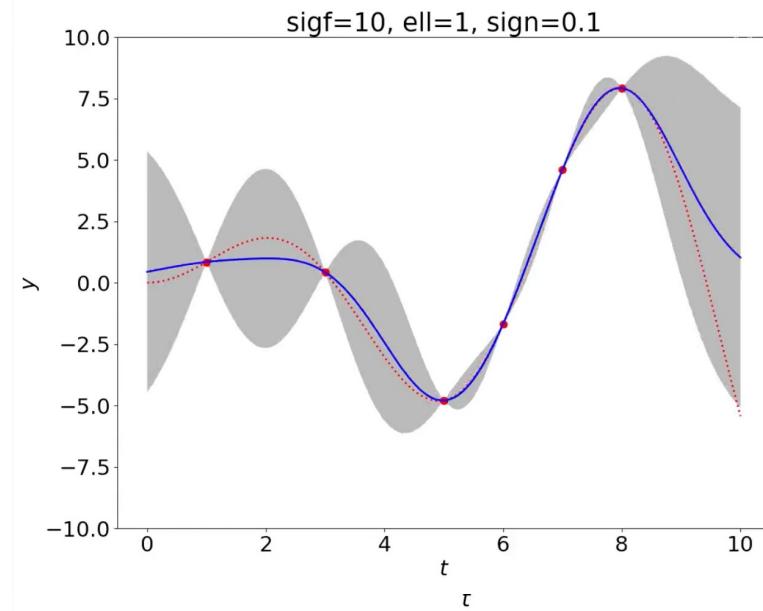
$$p(f|y) = \mathcal{N}(\mu_{f|y}, \Sigma_{f|y})$$

where:

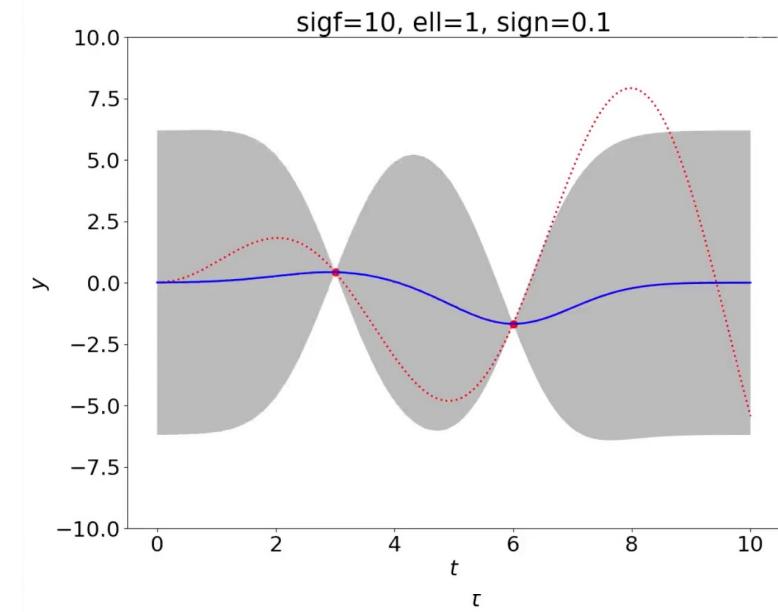
- $\mu_{f|y} = \mu_f + \Sigma_{fy}\Sigma_{yy}^{-1}(y - \mu_y)$
- $\Sigma_{f|y} = \Sigma_{ff} - \Sigma_{fy}\Sigma_{yy}^{-1}\Sigma_{fy}^T$

Now, we can plug in particular values for y , e.g.,

$$p(f|\mathbf{y}(t=6)) = \mathcal{N}(\mu_{f|\mathbf{y}(t=6)}, \Sigma_{f|\mathbf{y}(t=6)})$$



Example for GP regression



Example for GP regression

Remember the conditioning:

$$p(f|y) = \mathcal{N}(\mu_{f|y}, \Sigma_{f|y})$$

where:

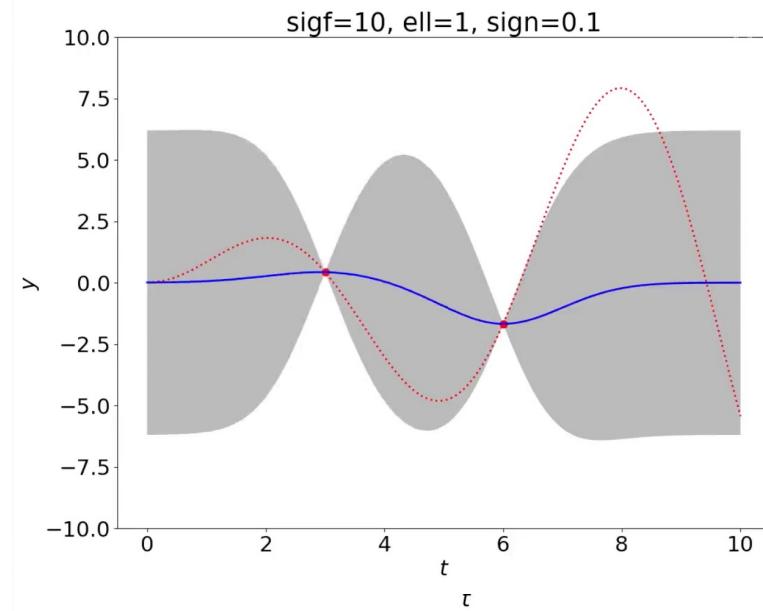
- $\mu_{f|y} = \mu_f + \Sigma_{fy}\Sigma_{yy}^{-1}(y - \mu_y)$
- $\Sigma_{f|y} = \Sigma_{ff} - \Sigma_{fy}\Sigma_{yy}^{-1}\Sigma_{fy}^T$

Remember the prediction:

$$p(y^*|y) = \mathcal{N}(\mu_{y^*|y}, \Sigma_{y^*|y})$$

where:

- $\mu_{y^*|y} = \mu_{y^*} + \Sigma_{y^*y}\Sigma_{yy}^{-1}(y - \mu_y)$
- $\Sigma_{y^*|y} = \Sigma_{y^*y} - \Sigma_{y^*y}\Sigma_{yy}^{-1}\Sigma_{y^*y}^T$



Example for GP regression

Remember the conditioning:

$$p(f|y) = \mathcal{N}(\mu_{f|y}, \Sigma_{f|y})$$

where:

- $\mu_{f|y} = \mu_f + \Sigma_{fy}\Sigma_{yy}^{-1}(y - \mu_y)$
- $\Sigma_{f|y} = \Sigma_{ff} - \Sigma_{fy}\Sigma_{yy}^{-1}\Sigma_{fy}^T$

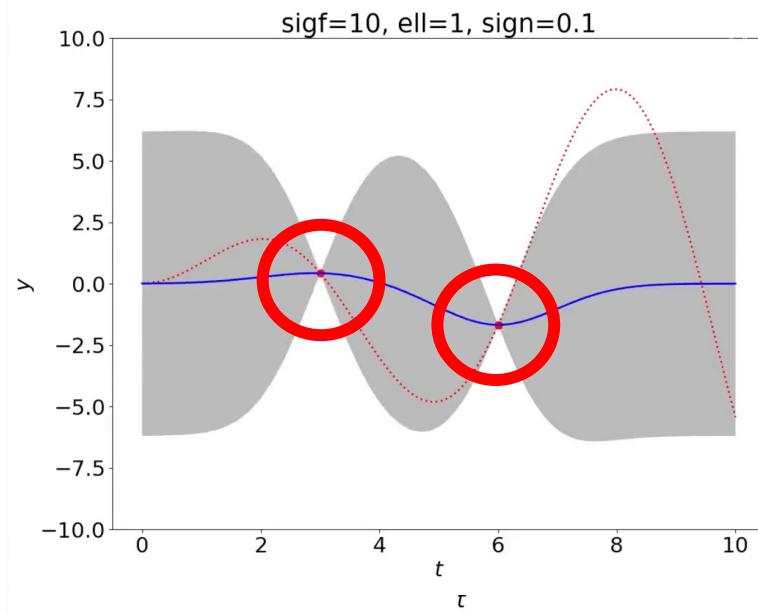
Remember the prediction:

$$p(y^*|y) = \mathcal{N}(\mu_{y^*|y}, \Sigma_{y^*|y})$$

where:

- $\mu_{y^*|y} = \mu_{y^*} + \Sigma_{y^*y}\Sigma_{yy}^{-1}(y - \mu_y)$
- $\Sigma_{y^*|y} = \Sigma_{y^*y} - \Sigma_{y^*y}\Sigma_{yy}^{-1}\Sigma_{y^*y}^T$

Test the prediction on $y = [t = 3 \atop t = 6]$



Example for GP regression

Remember the conditioning:

$$p(f|y) = \mathcal{N}(\mu_{f|y}, \Sigma_{f|y})$$

where:

- $\mu_{f|y} = \mu_f + \Sigma_{fy}\Sigma_{yy}^{-1}(y - \mu_y)$
- $\Sigma_{f|y} = \Sigma_{ff} - \Sigma_{fy}\Sigma_{yy}^{-1}\Sigma_{fy}^T$

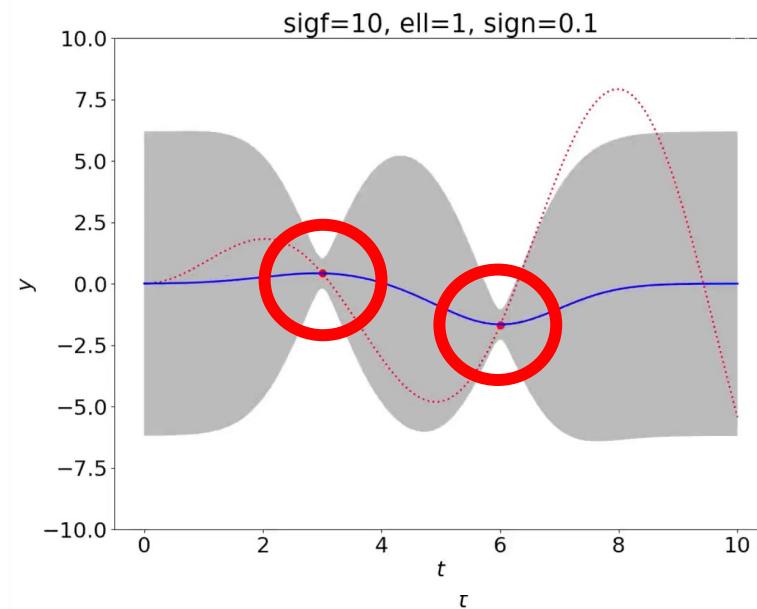
Remember the prediction:

$$p(y^*|y) = \mathcal{N}(\mu_{y^*|y}, \Sigma_{y^*|y})$$

where:

- $\mu_{y^*|y} = \mu_{y^*} + \Sigma_{y^*y}\Sigma_{yy}^{-1}(y - \mu_y)$
- $\Sigma_{y^*|y} = \Sigma_{y^*y} - \Sigma_{y^*y}\Sigma_{yy}^{-1}\Sigma_{y^*y}^T$

Test the prediction on $y = [t = 3 \quad t = 6]$



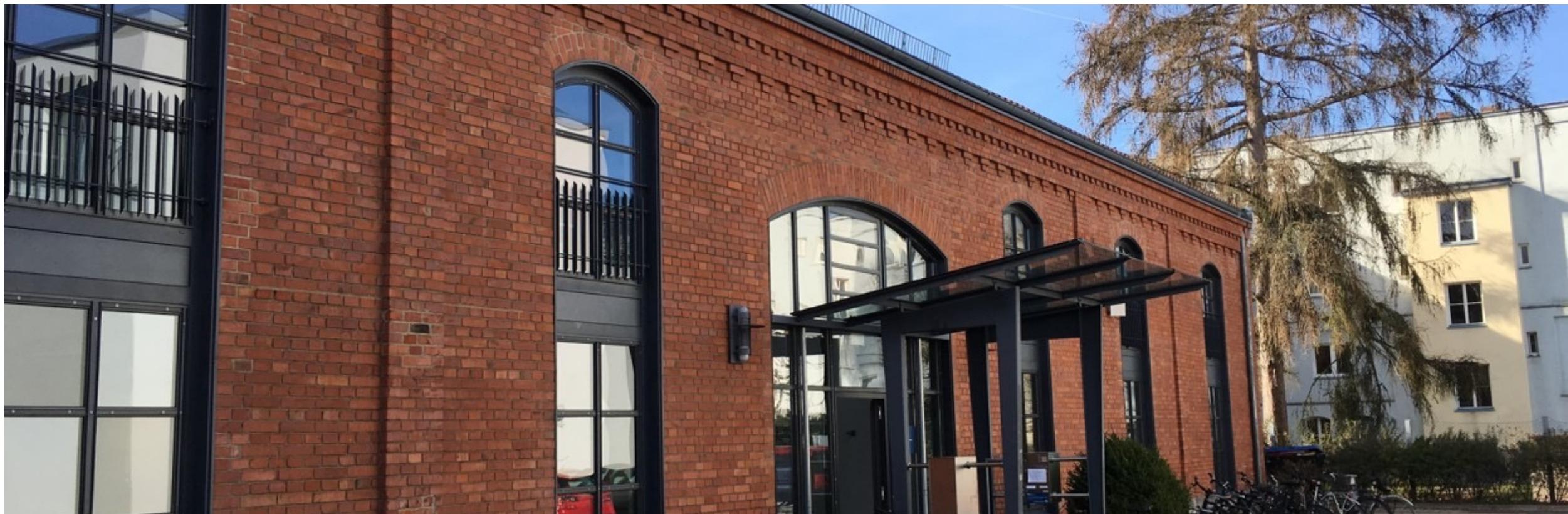


Lecture title

Recap



- Prior on functions
 - Recap of Gaussian distributions
 - Prior on parameters (indirect prior on functions)
- Gaussian processes
 - Multivariate Gaussian vs. GP
 - Definition
 - Example
- Gaussian process regression
 - Conditioning and inference
 - Prediction
 - Example



Machine Learning for Time Series

(MLTS or MLTS-Deluxe Lectures)

Dr. Dario Zanca

Machine Learning and Data Analytics (MaD) Lab
Friedrich-Alexander-Universität Erlangen-Nürnberg
25.10.2022

- Time series fundamentals and definitions (2 lectures)
- Bayesian Inference (1 lecture)
- Gaussian processes (2 lectures) ←
- State space models (2 lectures)
- Autoregressive models (1 lecture)
- Data mining on time series (1 lecture)
- Deep learning on time series (4 lectures)
- Domain adaptation (1 lecture)

In this lecture...

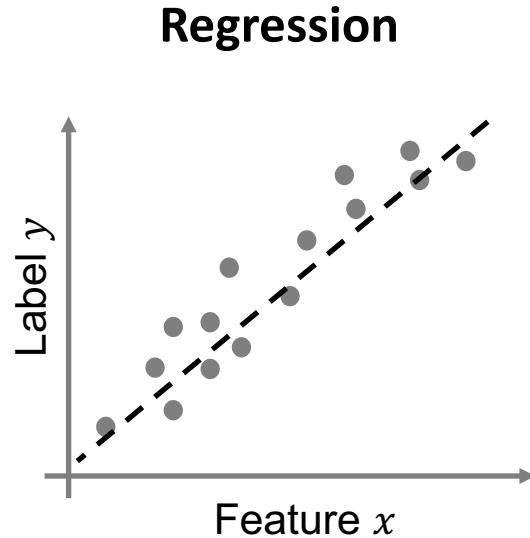
1. Gaussian process classification (GPC) formulation
2. Gaussian process classification (GPC) prediction



Gaussian process classification (GPC)

GPC formulation

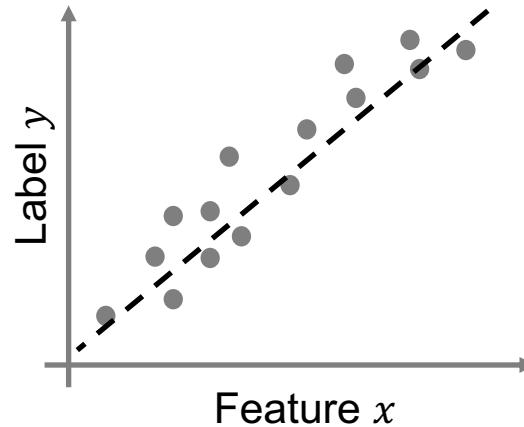




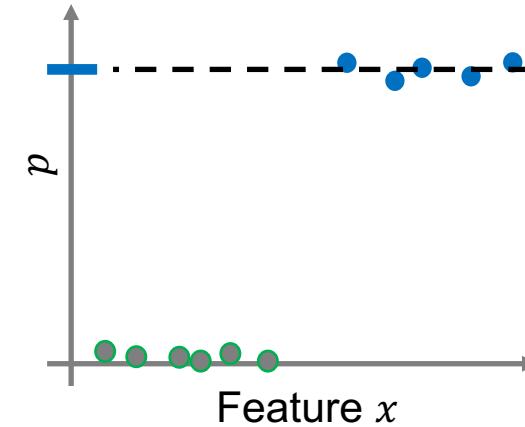
For regression we typically have:

- $x \in \mathbb{R}^d$
- $y_R \in \mathbb{R}$
- $y_R = f(x)$

Regression



Classification



For regression we typically have:

- $x \in \mathbb{R}^d$
- $y_R \in \mathbb{R}$
- $y_R = f(x)$

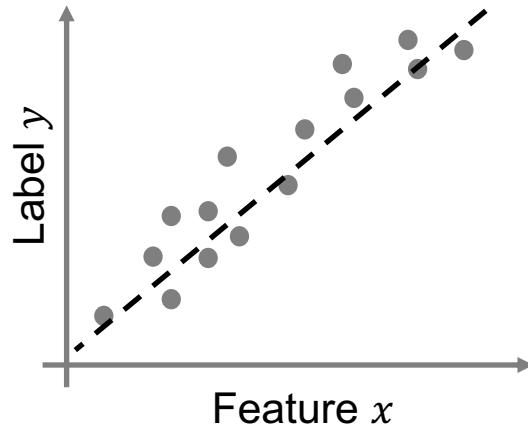
For (binary) classification, instead:

- $x \in \mathbb{R}^d$
- Task: $y_C \in \{-1, +1\}$
- $p \in [0, 1]$

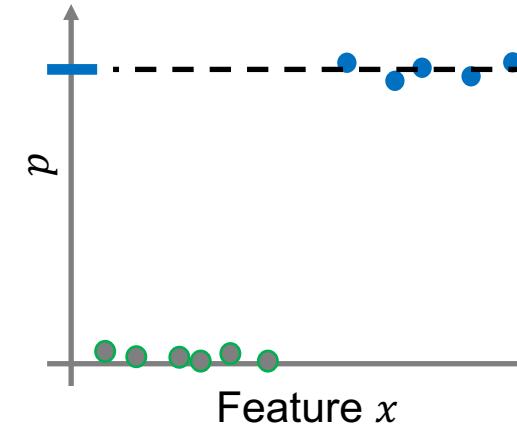
Gaussian Process Classification (GPC)

Regression vs. Classification

Regression



Classification



For regression we typically have:

- $x \in \mathbb{R}^d$
- $y_R \in \mathbb{R}$
- $y_R = f(x)$

For (binary) classification, instead:

- $x \in \mathbb{R}^d$
- Task: $y_C \in \{-1, +1\}$
- $p \in [0, 1]$

We use a Gaussian linear model in order to obtain the likelihood:

$$p(y = \pm 1 | x, w) = \sigma(x^T w)$$

where $\sigma(x^T w)$ is called sigmoid function.

We use a Gaussian linear model in order to obtain the likelihood:

$$p(y = \pm 1 | x, w) = \sigma(x^T w)$$

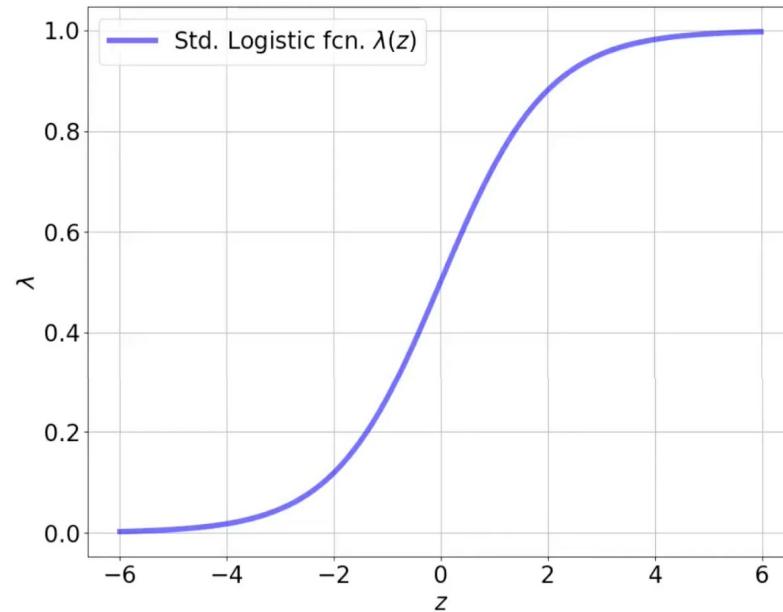
where $\sigma(x^T w)$ is called sigmoid function.

Notice:

- $p(y = \pm 1 | x, w)$ is the likelihood.
- Generally, we denote $\pi(x) := \sigma(x^T w)$

The sigmoid function

Common options for the sigmoid functions:

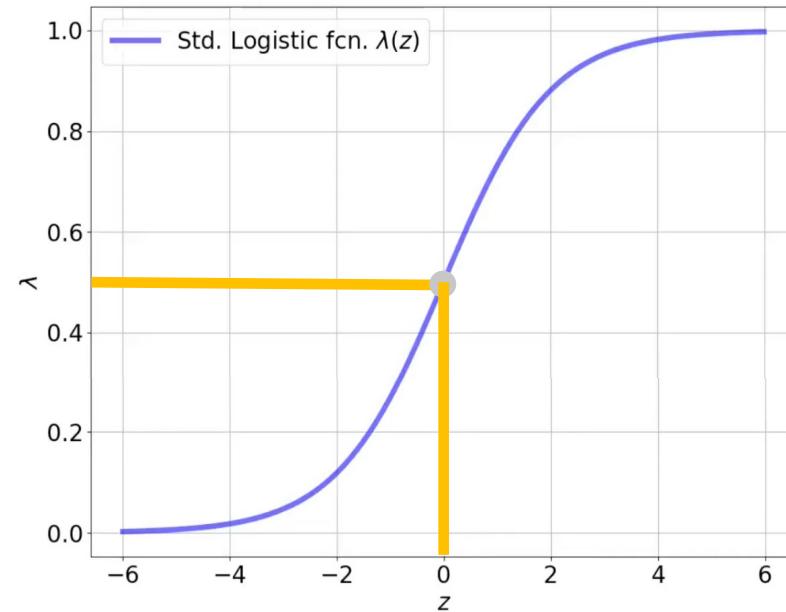


$$\lambda(z) = \frac{1}{1+e^{-z}}$$

(Logistic function)

The sigmoid function

Common options for the sigmoid functions:

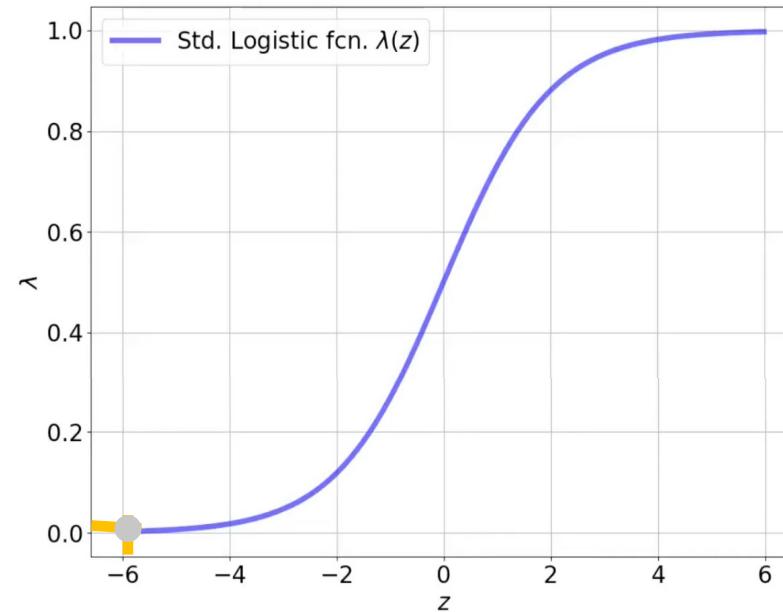


$$\lambda(z) = \frac{1}{1+e^{-z}}$$

(Logistic function)

The sigmoid function

Common options for the sigmoid functions:

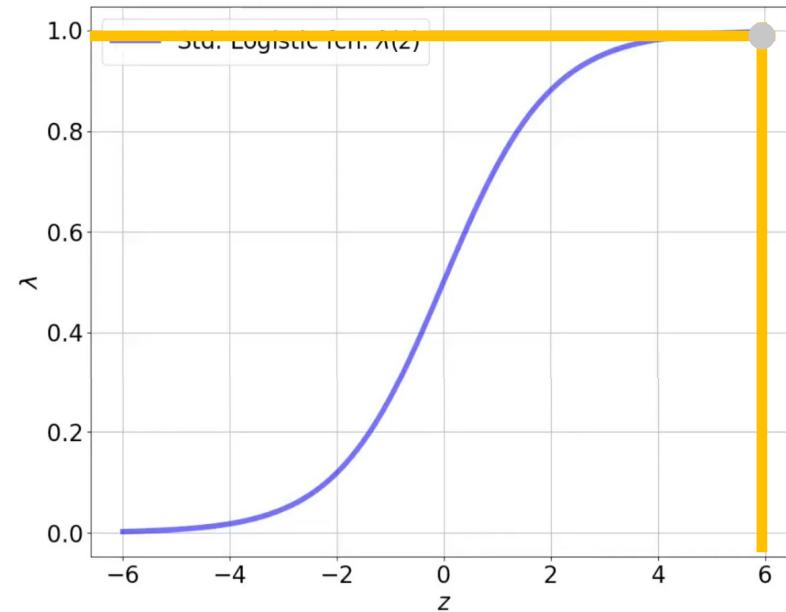


$$\lambda(z) = \frac{1}{1+e^{-z}}$$

(Logistic function)

The sigmoid function

Common options for the sigmoid functions:



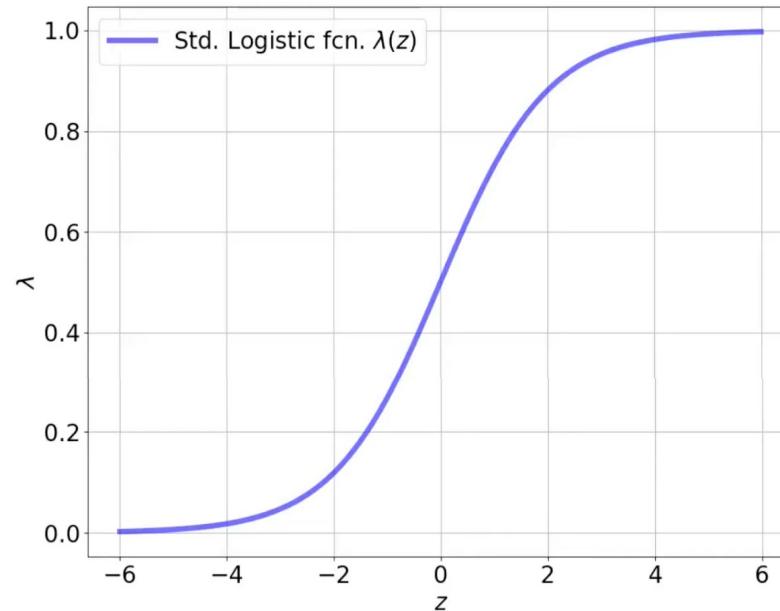
$$\lambda(z) = \frac{1}{1+e^{-z}}$$

(Logistic function)

Gaussian Process Classification (GPC)

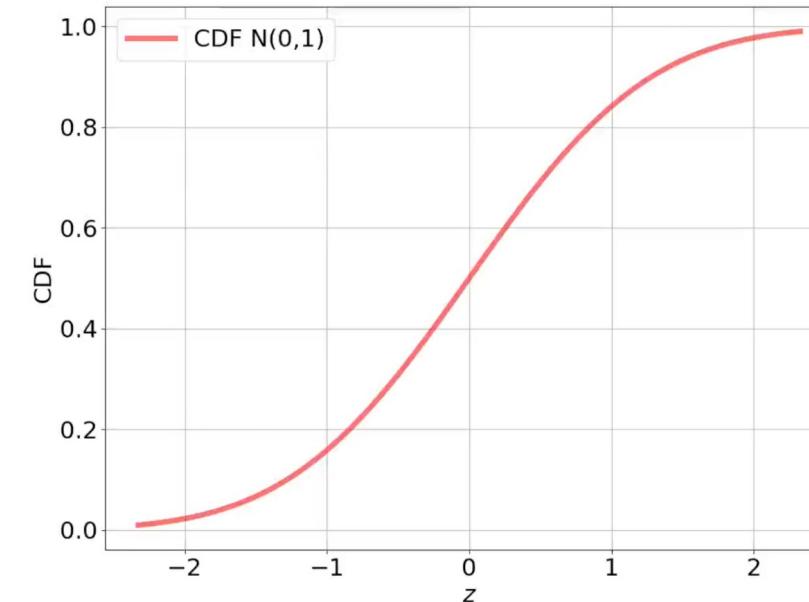
The sigmoid function

Common options for the sigmoid functions:



$$\lambda(z) = \frac{1}{1+e^{-z}}$$

(Logistic function)



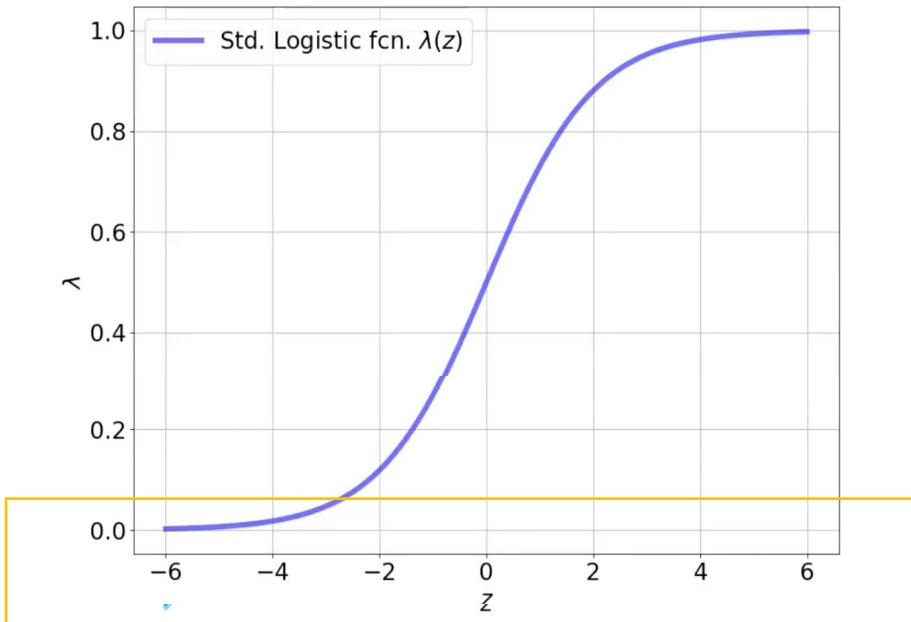
$$\phi(z) = \int_{-\infty}^z \mathcal{N}(x|0, 1) dz$$

(Cumulative distribution function - CDF)

Gaussian Process Classification (GPC)

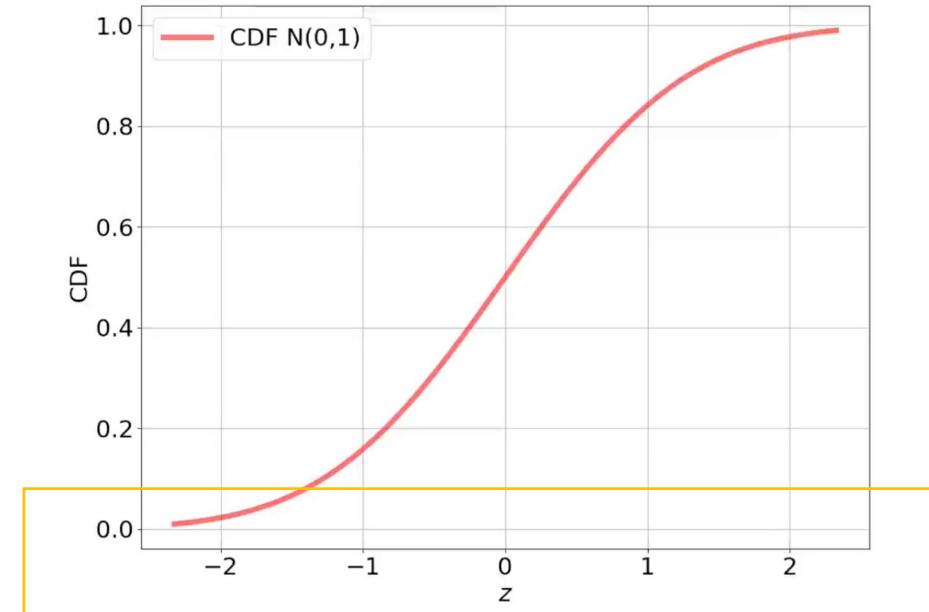
The sigmoid function

Common options for the sigmoid functions:



$$\lambda(z) = \frac{1}{1+e^{-z}}$$

(Logistic function)



$$\phi(z) = \int_{-\infty}^z \mathcal{N}(x|0, 1) dz$$

(Cumulative distribution function - CDF)

For a 2-class problem, we can write the likelihood of the value pair (x_i, y_i) :

$$\rightarrow \sigma(x_i^T w) \quad \text{if } y_i = +1$$

$$\rightarrow 1 - \sigma(x_i^T w) \quad \text{if } y_i = -1$$

For a 2-class problem, we can write the likelihood of the value pair (x_i, y_i) :

$$\rightarrow \sigma(x_i^T w) \quad \text{if } y_i = +1$$

$$\rightarrow 1 - \sigma(x_i^T w) \quad \text{if } y_i = -1$$

For symmetric sigmoid functions: $\sigma(-z) = 1 - \sigma(z)$

Thus: $p(y_i | x_i^T w) = \sigma(y_i f_i)$

- $y_i = \pm 1$ **(Sign)**
- $f_i = f(x_i) = x_i^T w$ **(Gaussian Process)**

Let's assume the prior on w :

$$w \sim \mathcal{N}(0, \sigma_p) \quad \text{or} \quad w \sim \mathcal{N}(0, \Sigma_p)$$

Then, we can write the posterior over weights:

$$p(w|y, X) = \frac{p(y|X, w) p(w)}{p(y|X)}$$

The marginal likelihood can be written as:

$$p(y|X) = \int p(y|X, w) p(w) dw$$

Step 1: Gaussian Process (GP) over latent function $f(x)$

Step 2: Filter f through a sigmoid function to obtain

$$\pi(x) = p(y = +1|x) = \sigma(f(x))$$



Gaussian process classification GPC prediction



Predict a new point x^* :

$$p(y^* = +1|x^*, D) = \int p(y^* = +1|w, x^*) \ p(w|D) \ dw$$

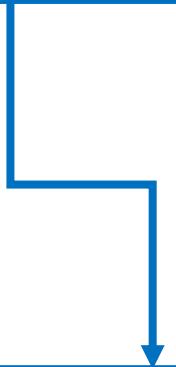
Predict a new point x^* :

$$p(y^* = +1|x^*, D) = \int p(y^* = +1|w, x^*) p(w|D) dw$$

Prediction = $\sigma(x^T w)$

Predict a new point x^* :

$$p(y^* = +1|x^*, D) = \int p(y^* = +1|w, x^*) p(w|D) dw$$



Unimodal Non-Gaussian

Prediction = $\sigma(x^T w)$

Step 1: Compute the distribution of f^* at case x^* .

$$p(f^*|X, y, x^*) = \int p(f^*|X, x^*, f) \ p(f|X, y) \ df$$

The posterior on $f(x)$ can be written as:

$$p(f|X, y) = \frac{p(y|f) \ p(f|X)}{p(y|X)}$$

Step 1: Compute the distribution of f^* at case x^* .

$$p(f^*|X, y, x^*) = \int p(f^*|X, x^*, f) \ p(f|X, y) \ df$$

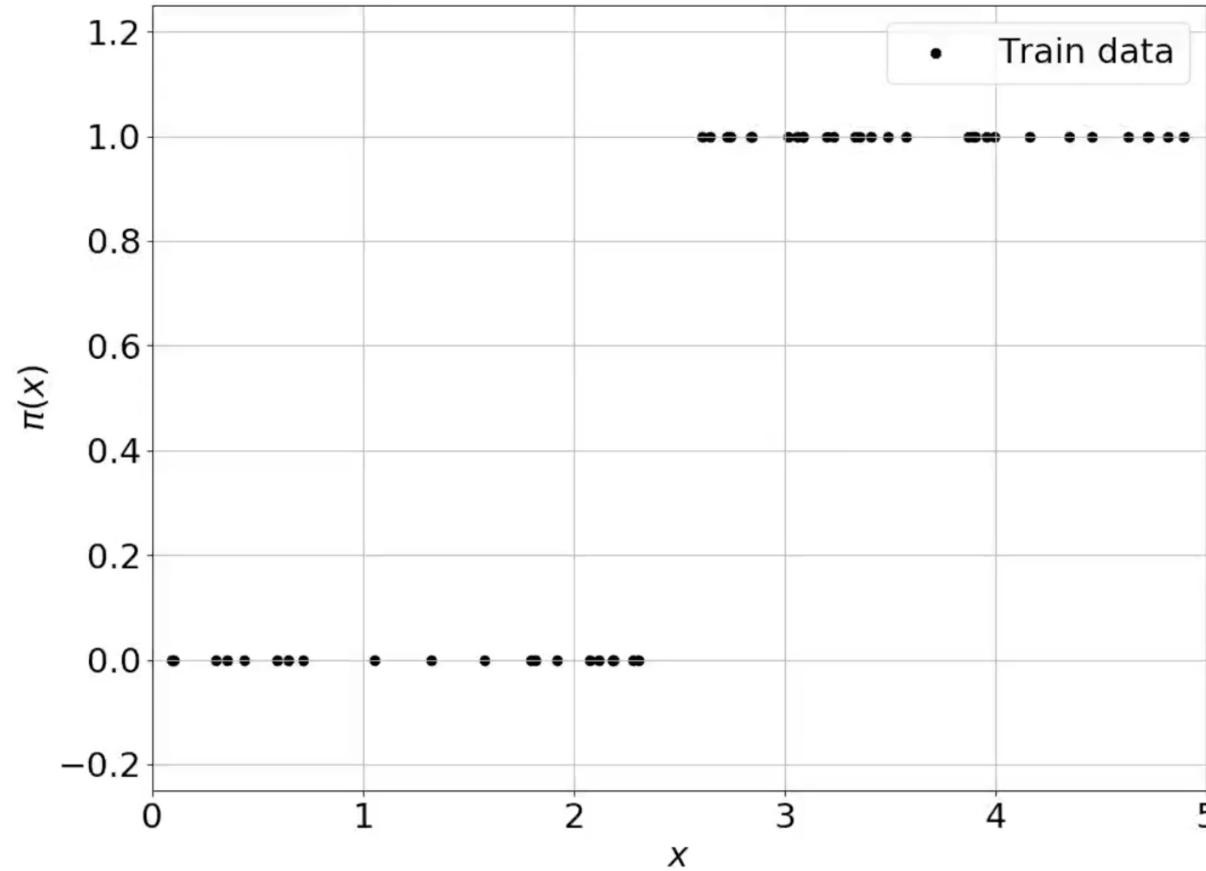
The posterior on $f(x)$ can be written as:

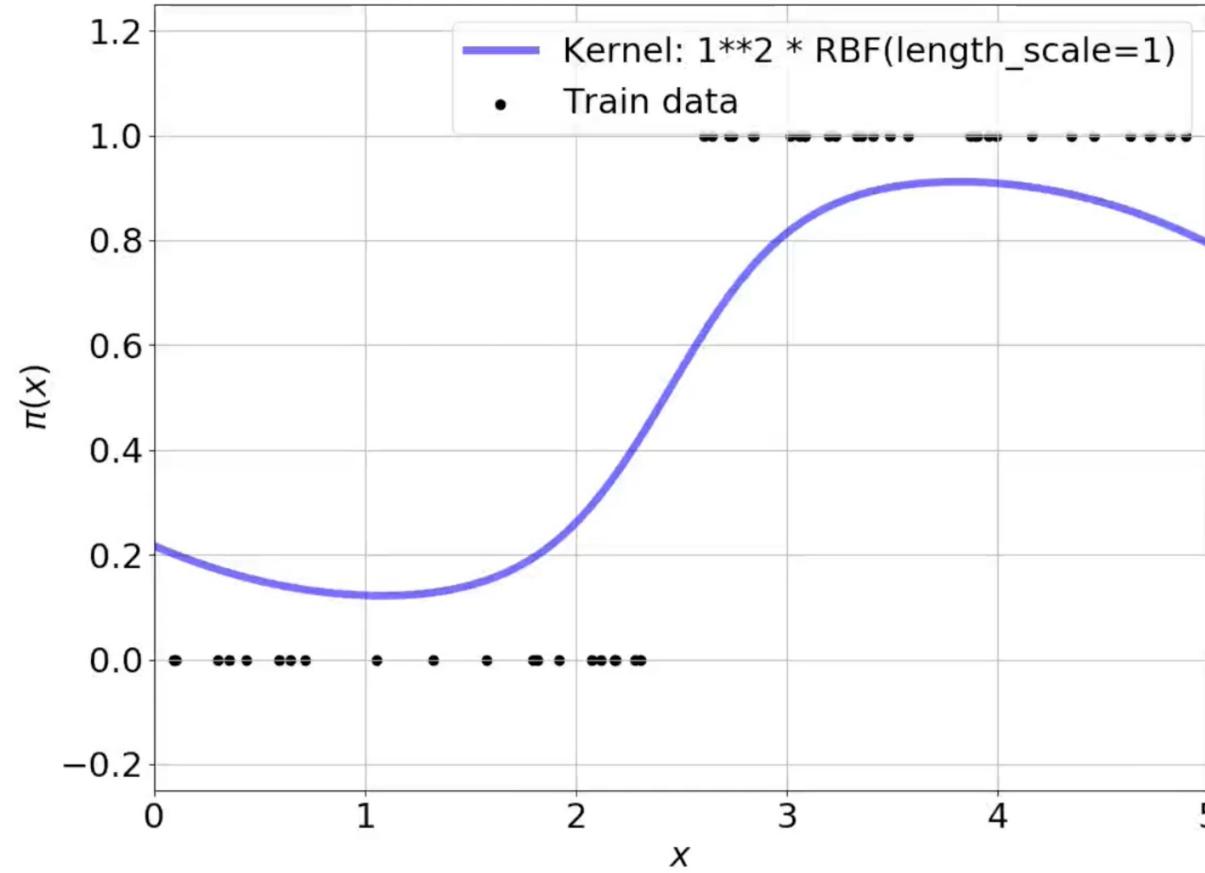
$$p(f|X, y) = \frac{p(y|f) \ p(f|X)}{p(y|X)}$$

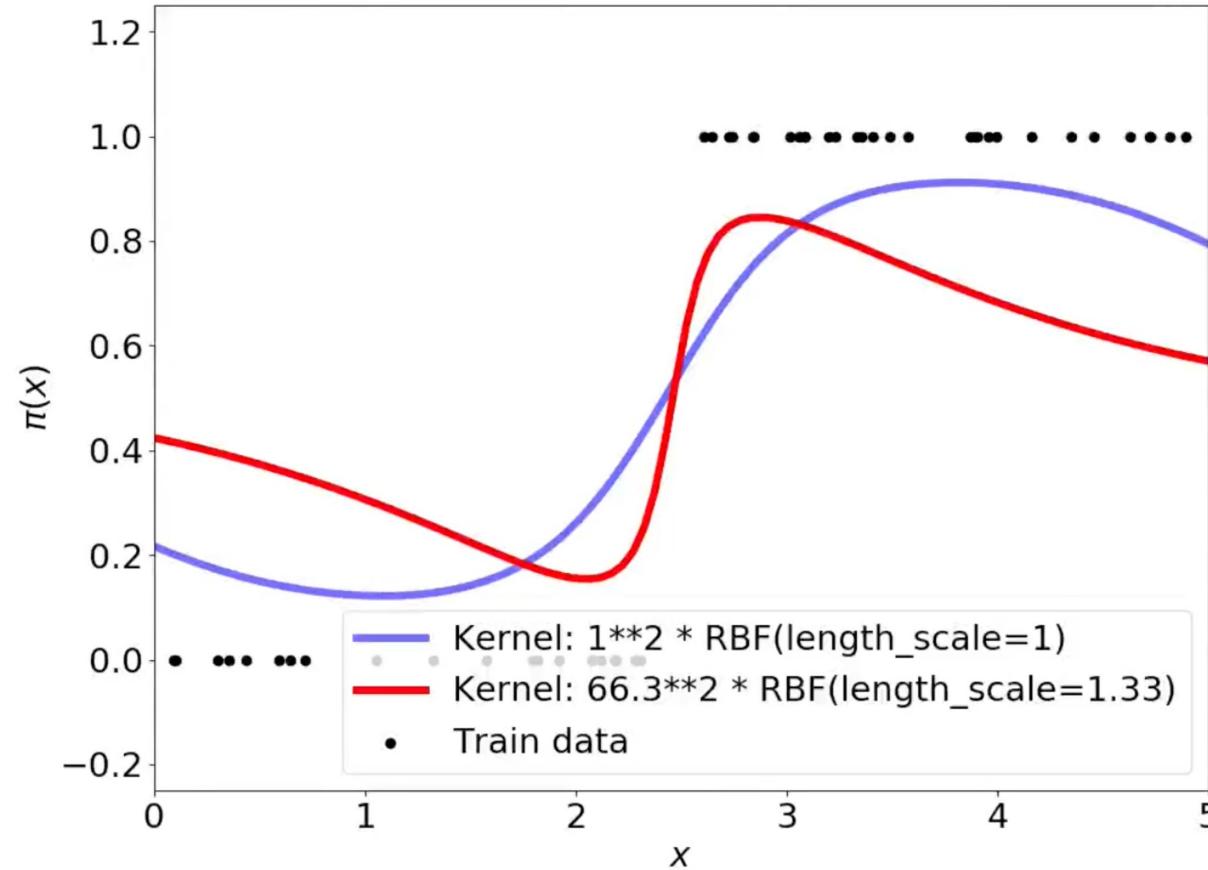
Step 2: Produce a probabilistic prediction π^* .

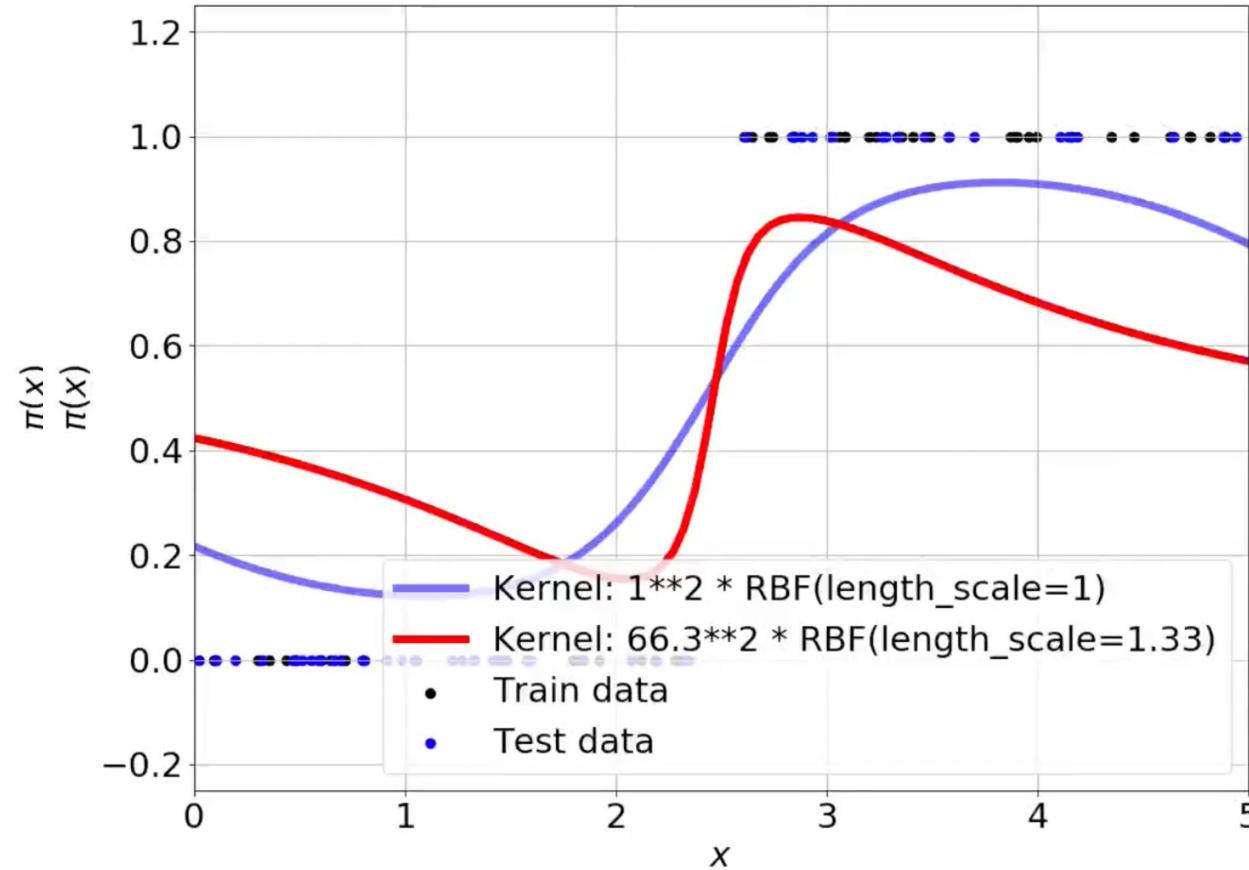
$$\pi^* \triangleq p(y^* = +1 | X, y, x^*) = \int \sigma(f^*) \ p(f^* | X, y, x^*) \ df^*$$

- $\pi^* = \pi(x^*)$ expresses the probability of the class
- The latent f has the role of nuisance function (we do not observe it)









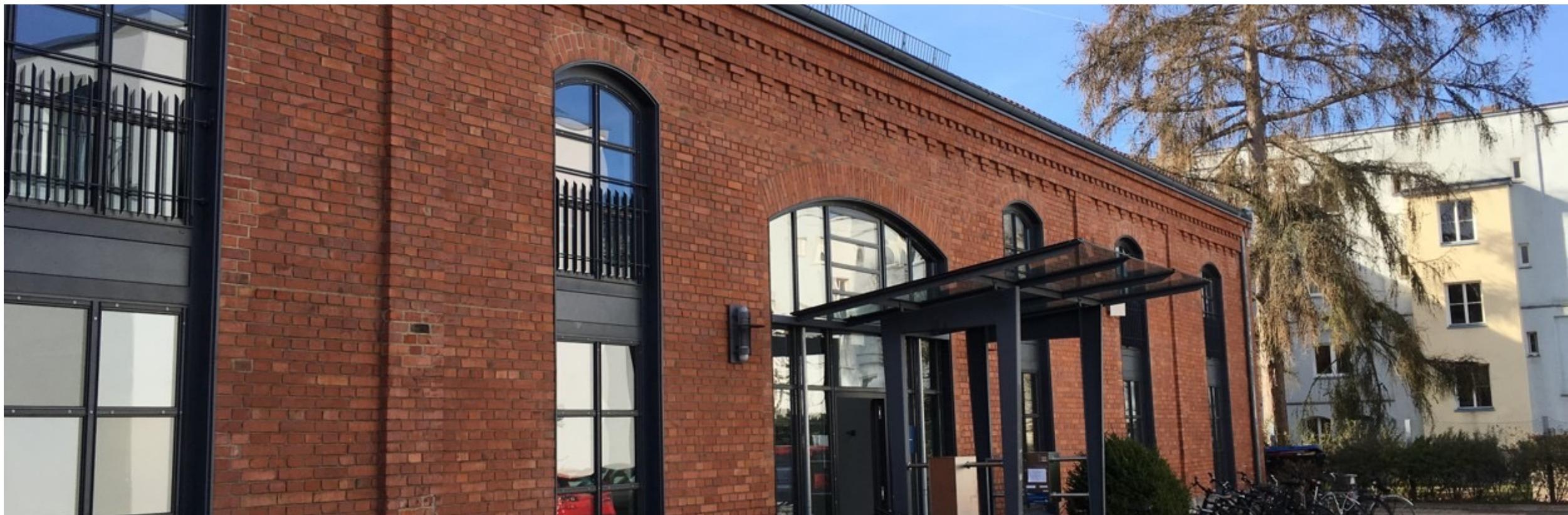


Gaussian process classification (GPC)

Recap



- Gaussian process classification (GPC) formulation
 - GLM
 - Sigmoid
- Gaussian process classification (GPC) prediction
 - Two step process



Machine Learning for Time Series

(MLTS or MLTS-Deluxe Lectures)

Dr. Dario Zanca

Machine Learning and Data Analytics (MaD) Lab
Friedrich-Alexander-Universität Erlangen-Nürnberg
06.12.2022

-
- Time series fundamentals and definitions (2 lectures)
 - Bayesian Inference (1 lecture)
 - Gaussian processes (2 lectures)
 - State space models (2 lectures) ←
 - Autoregressive models (1 lecture)
 - Data mining on time series (1 lecture)
 - Deep learning on time series (4 lectures)
 - Domain adaptation (1 lecture)

In this lecture...

- 1. State Space Models (SSMs)**
- 2. Kalman Filtering (KF)**
- 3. Real-world example with KF**
- 4. Extended Kalman Filter (EKF)**
- 5. Unscented Kalman Filter (UKF)**



State Space Models (SSMs) and Kalman Filtering (KF)

State Space Models (SSMs)



State space models

State Space Models (SSM) are commonly used in a wide range of applications:

- Object tracking (e.g., pedestrians or vehicles in self driving cars)
- Navigation (e.g., GPS)
- Aerospace engineering
- Remote surveillance
- Finance

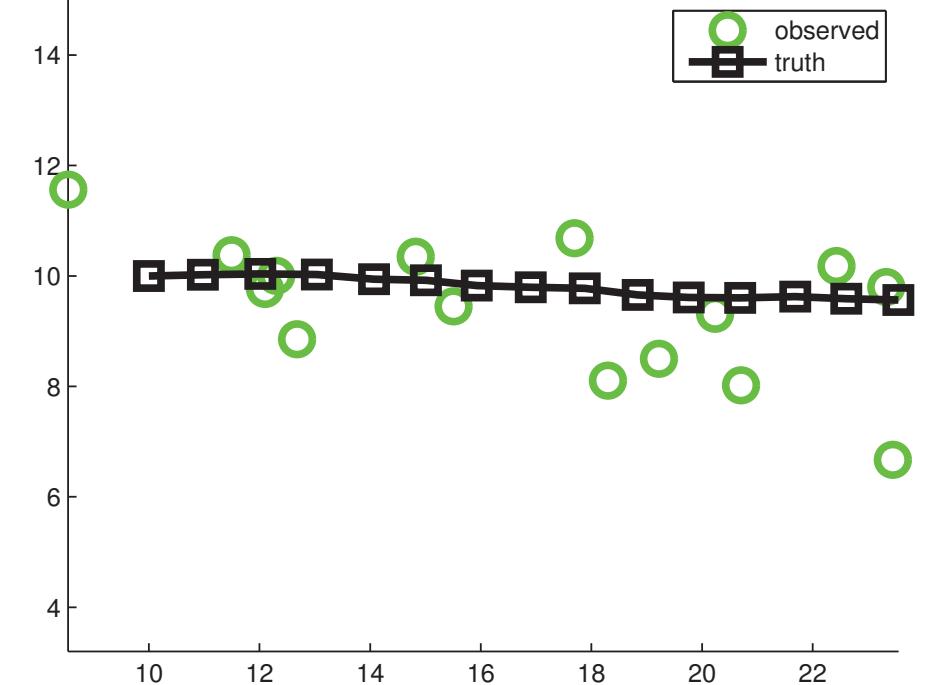


State Space Models

The SSM provides a general framework to describe deterministic and stochastic dynamical systems (i.e., **time varying systems**) which are indirectly observed through a stochastic process (i.e., **noisy measurements**).

It describes a probabilistic dependence between latent state variables and the observed measurements.

 The term “state space” originated in the area of control engineering (Kalman, 1960).



State Space Models

We denote with $z_n \in \mathbb{R}^D$ a continuous state variable at time n , and with $y_n \in \mathbb{R}^d$ the associated observation.

The state space model can be written in the **generic form**:

$$z_n = f(z_{n-1}, u_{n-1}, r_n) \quad \leftarrow \text{transition model}$$

$$y_n = h(z_n, u_n, q_n) \quad \leftarrow \text{measurement model}$$

where u_n is a deterministic (optional) variable, r_n is the system noise, and q_n is the observation noise.

We use SSM to **recursively** estimate the belief state and an initial state z_1 needs to be specified.

Linear-Gaussian State Space Models

Linear-Gaussian state space models (LG-SSM), also called linear dynamical systems, is an important special case of an SSM where we assume:

- The transition and the observation models are linear functions
 - $f(z_{n-1}, r_n) = Fz_{n-1} + r_n, F \in \mathbb{R}^{D \times D}$
 - $h(z_n, q_n) = Hz_n + q_n, H \in \mathbb{R}^{d \times D}$
- The system and observation noise processes are Gaussian
 - $r_n \sim \mathcal{N}(0, R)$
 - $q_n \sim \mathcal{N}(0, Q)$

We assume f , h and the noise processes to be known.

Linear-Gaussian State Space Models

The LG-SSM can be reformulated as:

Transition density: $p(z_n|z_{n-1}) = \mathcal{N}(Fz_{n-1}, R)$

Observation density: $p(y_n|z_n) = \mathcal{N}(Hz_n, Q)$

A general formulation of our problem:

- We are interested to have an estimation of our hidden state at time n .
- We estimate hidden states by a density.

We can analytically compute Kalman filtering for linear functions and Gaussian noises.

Linear-Gaussian State Space Models

The conditional mean is a good candidate for estimating the state z_n :

$$\bar{\mu}_n = \mathbb{E}[z_n | y_{1:k}]$$

A suitable measure for the uncertainty of the hidden state z_n is, then, given by the conditional covariance:

$$\bar{\Sigma}_n = \mathbb{E}[(z_n - \bar{\mu}_n)(z_n - \bar{\mu}_n)^T | y_{1:k}]$$

Depending on the value of k , we call the problem:

- Prediction, if $k < n$
- Filtering, if $k = n$
- Smoothing, if $k > n$



State Space Models (SSMs) and Kalman Filtering (KF)

Kalman Filtering (KF)



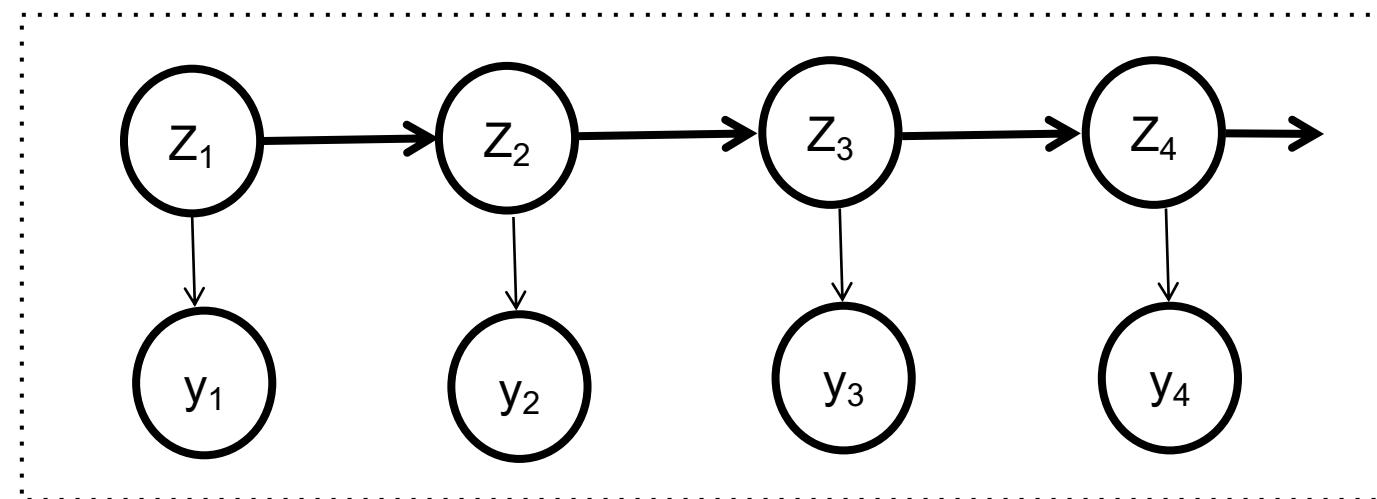
Review concept: the Markov property

When dealing with sequential data, the Markov property ensures that each data point **depends only on the previous data point** (and not to older instances!).

In formulas:

$$p(z_n | z_{n-1}, y_{1:n-1}) = p(z_n | z_{n-1})$$

$$p(y_n | z_n, y_{1:n-1}) = p(y_n | z_n)$$



Kalman filtering (KF)

The Kalman filtering is an algorithm for exact Bayesian filtering for linear-Gaussian state space models.

- In other words, we recursively estimate the state of a dynamical system
- E.g., indirect measurements of a rocket thruster temperature

It consists of two steps:

1. **Prediction:** Given an initial state we leverage our knowledge of the process to produce an estimate of the current state, along with its uncertainty
2. **Correction (or Filtering):** We update the current belief based on new measurements (sensory information)

Since everything is Gaussian, we can perform the prediction and update steps in closed form.

Kalman filtering (KF)

The predictive density (or prior) is given by:

$$\begin{aligned}
 p(z_n | y_{1:n-1}) &= \int p(z_n, z_{n-1} | y_{1:n-1}) dz_{n-1} \\
 &= \int p(z_n | z_{n-1}, y_{1:n-1}) p(z_{n-1} | y_{1:n-1}) dz_{n-1} \\
 &= \underbrace{\int p(z_n | z_{n-1})}_{\text{transition density}} \underbrace{p(z_{n-1} | y_{1:n-1})}_{\text{filtering density}} dz_{n-1}
 \end{aligned}$$



Markov property

We can compute the filtering density (or posterior) using the Bayes rule:

$$\begin{aligned}
 p(z_n | y_{1:n}) &\propto p(y_n | z_n, y_{1:n-1}) p(z_n | y_{1:n-1}) \\
 &\propto \underbrace{p(y_n | z_n)}_{\text{likelihood}} \underbrace{p(z_n | y_{1:n-1})}_{\text{prior}}
 \end{aligned}$$



Markov property

Integrals are analytically-tractable for Kalman filtering for an LG-SSM.

Kalman filtering (KF)

The Kalman filter is only concerned with propagating the first two moments (mean and variance) of the filtering density.

Kalman filtering (KF)

The Kalman filter is only concerned with propagating the first two moments (mean and variance) of the filtering density.

We assume the filtering density at time $n - 1$ is given by

$$p(z_n | y_{1:n-1}) = \mathcal{N}(\bar{\mu}_{n-1}, \bar{\Sigma}_{n-1})$$

Kalman filtering (KF)

The Kalman filter is only concerned with propagating the first two moments (mean and variance) of the filtering density.

We assume the filtering density at time $n - 1$ is given by

$$p(z_{n-1} | y_{1:n-1}) = \mathcal{N}(\bar{\mu}_{n-1}, \bar{\Sigma}_{n-1})$$

Then, the predictive density is Gaussian:

$$\begin{aligned} p(z_n | y_{1:n-1}) &= \int \mathcal{N}(Fz_{n-1}, R) \mathcal{N}(\bar{\mu}_{n-1}, \bar{\Sigma}_{n-1}) dz_{n-1} \\ &\quad \text{transition density} \quad \text{filtering density} \\ &= \mathcal{N}(F\bar{\mu}_{n-1}, R + F\bar{\Sigma}_{n-1}F^T) \\ &= \mathcal{N}(\hat{\mu}_n, \hat{\Sigma}_n) \end{aligned}$$

Kalman filtering (KF)

The new filtering density is also Gaussian:

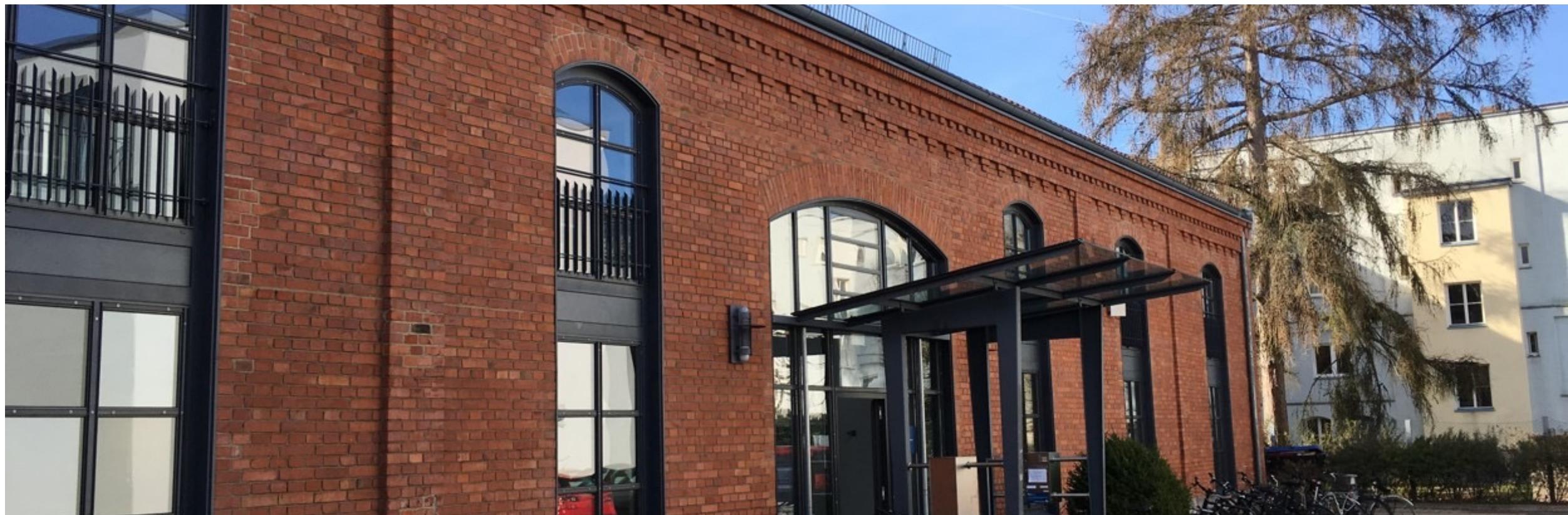
$$\begin{aligned} p(z_n | y_{1:n}) &\propto \underbrace{\mathcal{N}(Hz_n, Q)}_{\text{likelihood}} \underbrace{\mathcal{N}(\hat{\mu}_n, \hat{\Sigma}_n)}_{\text{predictive density}} \\ &= \mathcal{N}(\hat{\mu}_n + K_n(y_n - H\hat{\mu}_n), (I - K_n H)\hat{\Sigma}_n) \\ &= \mathcal{N}(\bar{\mu}_n, \bar{\Sigma}_n) \end{aligned}$$

Where K_n is the Kalman gain matrix:

$$K_n = \hat{\Sigma}_n H^T S_n^{-1}$$

$$S_n = H \hat{\Sigma}_n H^T + Q_n$$

We see that filtering and predictive densities in KF are Gaussian at any time



State Space Models (SSMs) and Kalman Filtering (KF)

Real-world example with KF



KF example: biker position estimation



x is the position of a rider
 \dot{x} is the rider's velocity

$$z = \begin{bmatrix} x \\ \dot{x} \end{bmatrix}$$
 is the system's state

- We measure the distance of a biker from an antenna on a 1-dimensional plane.
- From the antenna we get noisy observations about the position and the velocity of the biker.

KF example: biker position estimation



x is the position of a rider
 \dot{x} is the rider's velocity

$$z = \begin{bmatrix} x \\ \dot{x} \end{bmatrix}$$

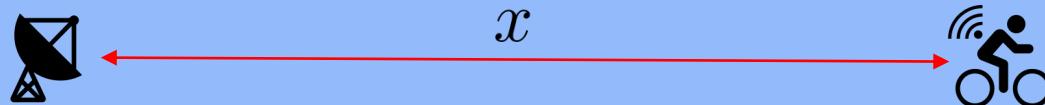
is the system's state

Transition model:

$$x_n = x_{n-1} + \dot{x}_{n-1} \Delta t + \frac{1}{2} \frac{f}{m} (\Delta t)^2 + r_{1_n}$$

$$\dot{x}_n = \dot{x}_{n-1} + \frac{f}{m} \Delta t + r_{2_n}$$

KF example: biker position estimation



x is the position of a rider
 \dot{x} is the rider's velocity

$$z = \begin{bmatrix} x \\ \dot{x} \end{bmatrix} \quad \text{is the system's state}$$

Transition model:

$$x_n = x_{n-1} + \dot{x}_{n-1}\Delta t + \frac{1}{2} \frac{f}{m} (\Delta t)^2 + r_{1_n}$$

$$\dot{x}_n = \dot{x}_{n-1} + \frac{f}{m} \Delta t + r_{2_n}$$

$$\begin{bmatrix} x_n \\ \dot{x}_n \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_{n-1} \\ \dot{x}_{n-1} \end{bmatrix} + \begin{bmatrix} \frac{(\Delta t)^2}{2m} \\ \frac{f}{m} \Delta t \end{bmatrix} f + \begin{bmatrix} r_{1_n} \\ r_{2_n} \end{bmatrix}$$

$$z_n = F z_{n-1} + B u_n + r_n$$

r_n Gaussian Noise

F State Transition Matrix

B Additional Information

KF example: biker position estimation



x is the position of a rider
 \dot{x} is the rider's velocity

$$z = \begin{bmatrix} x \\ \dot{x} \end{bmatrix}$$

is the system's state

Measurement model:

$$y_n = x_n + q_n$$

$$y_n = [1 \quad 0] \begin{bmatrix} x_n \\ \dot{x}_n \end{bmatrix} + \begin{bmatrix} q_{1n} \\ q_{2n} \end{bmatrix}$$

KF example: biker position estimation



x is the position of a rider
 \dot{x} is the rider's velocity

$$z = \begin{bmatrix} x \\ \dot{x} \end{bmatrix}$$

is the system's state

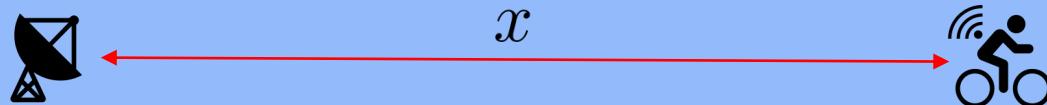
Measurement model:

$$y_n = x_n + q_n$$

$$y_n = [1 \quad 0] \begin{bmatrix} x_n \\ \dot{x}_n \end{bmatrix} + \begin{bmatrix} q_{1n} \\ q_{2n} \end{bmatrix}$$

$$y_n = Hz_n + q_n$$

KF example: biker position estimation



x is the position of a rider
 \dot{x} is the rider's velocity

$$z = \begin{bmatrix} x \\ \dot{x} \end{bmatrix} \quad \text{is the system's state}$$

Initial conditions:

$$z_0 = \begin{bmatrix} x_0 \\ \dot{x}_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 5 \end{bmatrix} \quad \Sigma_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

We also need to define covariance matrices associated with r and q:

$$R = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad Q = \begin{bmatrix} 3 & 0 \\ 0 & 3 \end{bmatrix}$$

Notice: In practice: we perform a search over these parameters.

KF: An algorithmic view

Prediction step (time update):

$$\bar{z}_n = F_n z_{n-1} + B_n u_n$$

$$\bar{\Sigma}_n = F_n \Sigma_{n-1} F_n^T + R_n$$

KF: An algorithmic view

Prediction step (time update):

$$\bar{z}_n = F_n z_{n-1} + B_n u_n$$

$$\bar{\Sigma}_n = F_n \Sigma_{n-1} F_n^T + R_n$$

Filtering step (Measurement update):

$$K_n = \bar{\Sigma}_n H_n^T (H_n \bar{\Sigma}_n H_n^T + Q_n)^{-1}$$

$$z_n = \bar{z}_n + K_n (y_n - H_n \bar{z}_n)$$

$$\Sigma_n = (1 - K_n H_n) \bar{\Sigma}_n$$

KF: An algorithmic view

Prediction step (time update):

$$\bar{z}_n = F_n z_{n-1} + B_n u_n$$

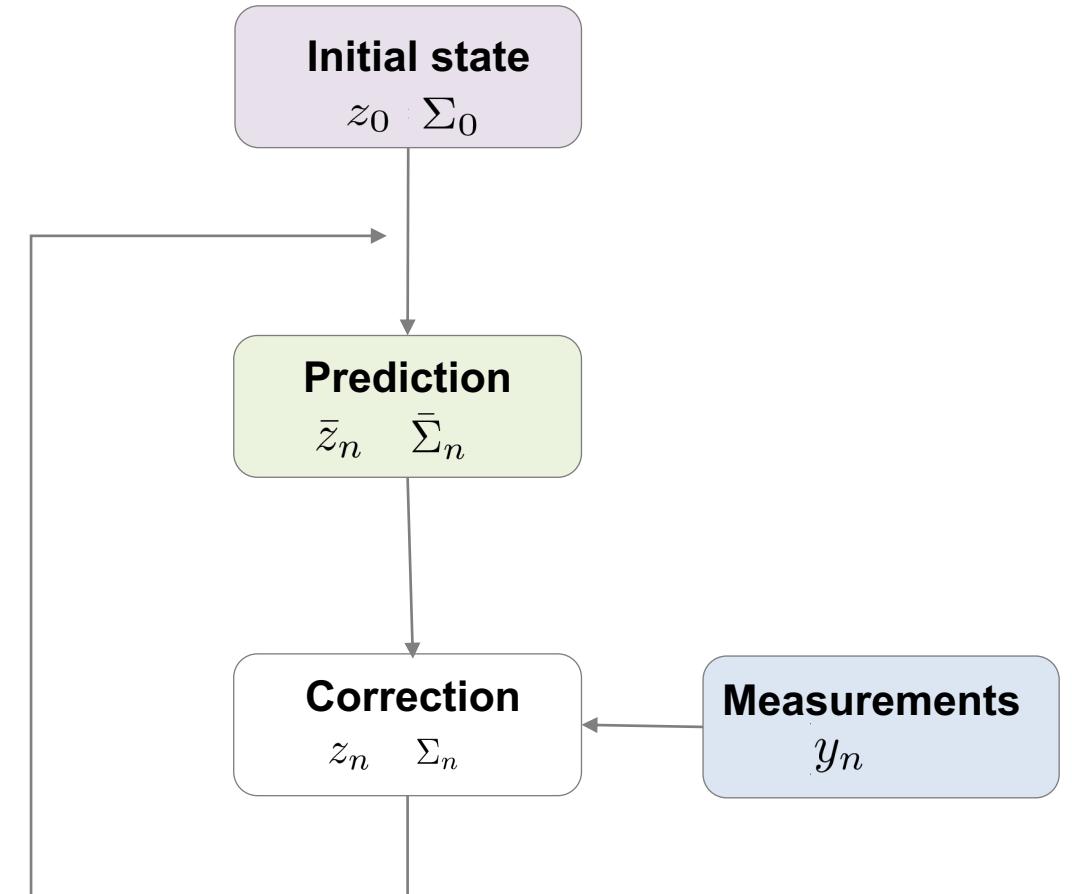
$$\bar{\Sigma}_n = F_n \Sigma_{n-1} F_n^T + R_n$$

Filtering step (Measurement update):

$$K_n = \bar{\Sigma}_n H_n^T (H_n \bar{\Sigma}_n H_n^T + Q_n)^{-1}$$

$$z_n = \bar{z}_n + K_n (y_n - H_n \bar{z}_n)$$

$$\Sigma_n = (1 - K_n H_n) \bar{\Sigma}_n$$





State Space Models (SSMs) and Kalman Filtering (KF)

Extended Kalman Filter (EKF)



Motivations

Recall KF assumptions:

- Linear state transition model
- Linear measurement model
- Gaussian noise

If these assumptions do not hold, we need apply other methods!

Linearized Dynamical Systems

When the transition model f and/or the measurement model h are not linear, then:

- the transition probability $p(z_n|z_{n-1})$ is non-Gaussian
- the predictive distribution $p(z_n|y_{1:n-1})$ is, in general, intractable

A possible approach is to consider the **linearized** dynamical system (constructed using the **Taylor expansion**) around the estimate of the current state:

$$z_n \approx f(\bar{\mu}_{n-1}) + \bar{F}_{n-1}(z_{n-1} - \bar{\mu}_{n-1}) + \dots + r_n$$

$$y_n \approx h(\hat{\mu}_{n-1}) + \hat{H}_n(z_n - \hat{\mu}_n) + \dots + q_n$$

where \bar{F} and \hat{H} are the Jacobian of f and h respectively, w.r.t z .

Linearized Dynamical Systems

Given a linearized system:

$$z_n \approx f(\bar{\mu}_{n-1}) + \bar{F}_{n-1}(z_{n-1} - \bar{\mu}_{n-1}) + \dots + r_n$$

$$y_n \approx h(\hat{\mu}_{n-1}) + \hat{H}_n(z_n - \hat{\mu}_n) + \dots + q_n$$

If we use the linear term in the Taylor expansion and discard the higher order parts, the approximated transition density and likelihood are again Gaussian:

$$q(\mathbf{z}_n | \mathbf{z}_{n-1}) = \mathcal{N}(\mathbf{f}(\bar{\mu}_{n-1}) + \bar{F}_{n-1}(\mathbf{z}_{n-1} - \bar{\mu}_{n-1}), \mathbf{R}),$$

$$q(\mathbf{y}_n | \mathbf{z}_n) = \mathcal{N}(\mathbf{h}(\hat{\mu}_n) + \hat{H}_n(\mathbf{z}_n - \hat{\mu}_n), \mathbf{Q}).$$

This idea is the basis for the so called Extended Kalman Filter (EKF).

Extended Kalman Filter (EKF)

Let's assume the filtering density is equal to $\mathcal{N}(\bar{\mu}_{n-1}, \Sigma_{n-1})$ at time $n - 1$.

The approximated predictive density is Gaussian:

$$\begin{aligned} p(\mathbf{z}_n | \mathbf{y}_{1:n-1}) &= \int q(\mathbf{z}_n | \mathbf{z}_{n-1}) p(\mathbf{z}_{n-1} | \mathbf{y}_{1:n-1}) d\mathbf{z}_{n-1} \\ &= \mathcal{N}\left(\underbrace{\mathbf{f}(\bar{\mu}_{n-1})}_{=\hat{\mu}_n}, \underbrace{\bar{\mathbf{F}}_{n-1} \bar{\Sigma}_{n-1} \bar{\mathbf{F}}_{n-1}^T + \mathbf{R}}_{=\hat{\Sigma}_n}\right). \end{aligned}$$

The approximated filtering density is also Gaussian:

$$\begin{aligned} p(\mathbf{z}_n | \mathbf{y}_{1:n}) &\propto q(\mathbf{y}_n | \mathbf{z}_n) p(\mathbf{z}_n | \mathbf{y}_{1:n-1}) \\ &= \mathcal{N}(\bar{\mu}_n, \bar{\Sigma}_n), \end{aligned}$$

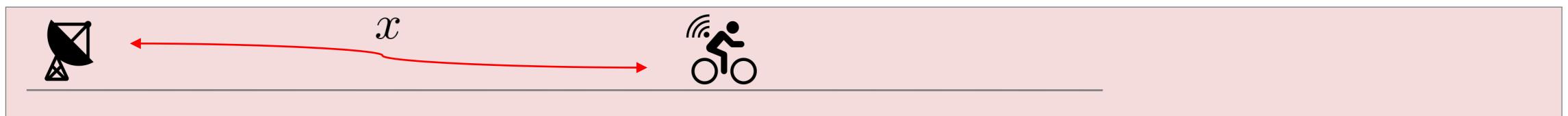
$$\begin{aligned} \bar{\mu}_n &= \hat{\mu}_n + \mathbf{K}_n (\mathbf{y}_n - \mathbf{h}(\hat{\mu}_n)), \\ \bar{\Sigma}_n &= (\mathbf{I} - \mathbf{K}_n \hat{\mathbf{H}}_n) \hat{\Sigma}_n, \\ \mathbf{K}_n &= \hat{\Sigma}_n \hat{\mathbf{H}}_n^T (\hat{\mathbf{H}}_n \hat{\Sigma}_n \hat{\mathbf{H}}_n^T + \mathbf{Q}_n)^{-1}. \end{aligned}$$

Example: EKF



$$z_n = F z_{n-1} + B u_n + r_n$$

$$y_n = H z_n + q_n$$



$$z_n = f(z_{n-1}, u_n) + r_n$$

$$y_n = h(z_n) + q_n$$

Example: EKF



$$z_n = f(z_{n-1}, u_n) + r_n$$

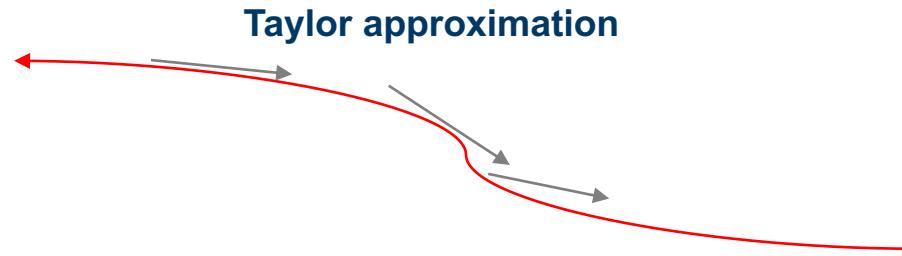
$$y_n = h(z_n) + q_n$$

Assumption: non-linear (but differentiable) transition and/or measurement models.

→ We apply first-order Taylor expansion:

$$J_{n-1}^f = \nabla f|_{z_{n-1}, u_n}$$

$$J_n^h = \nabla h|_{z_n}$$



This approach works if the functions are “sufficiently” linear (or locally linear).

EKF: An algorithmic view

Prediction step (Temporal update):

$$\bar{z}_n = f(z_{n-1}, u_n)$$

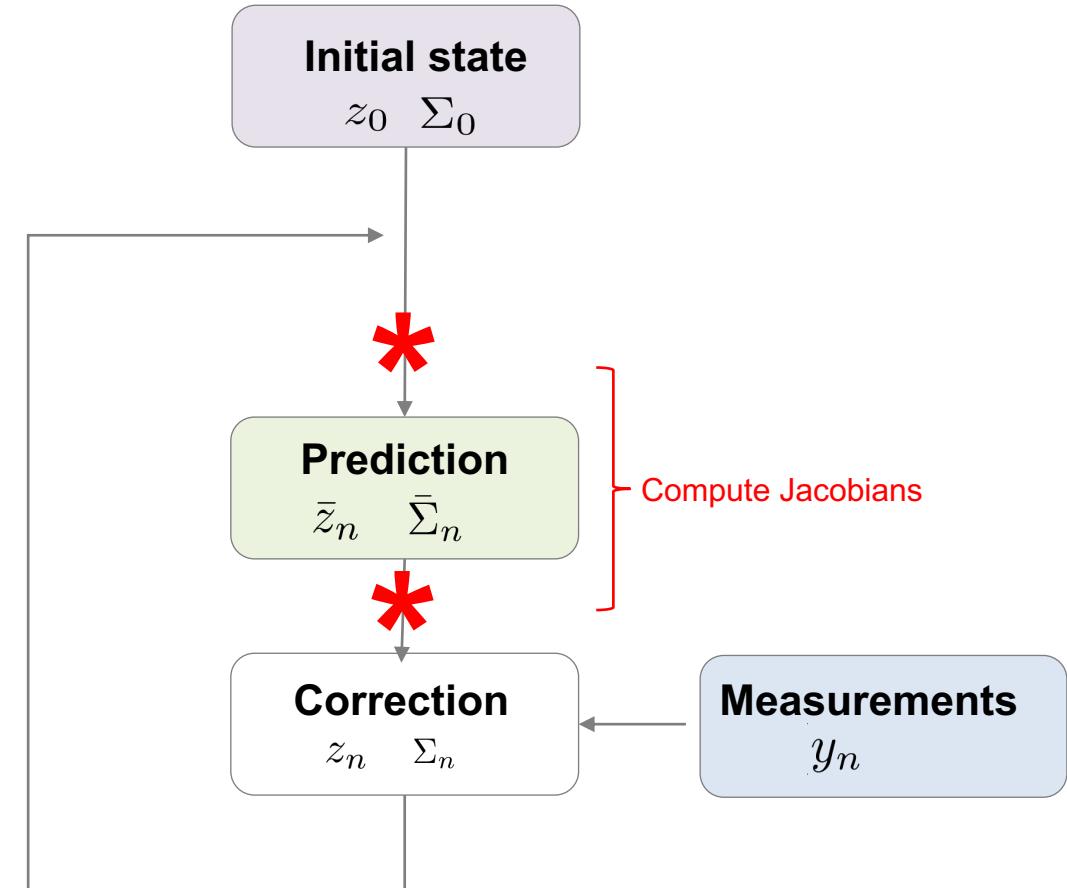
$$\bar{\Sigma}_n = J_n^f \Sigma_{n-1} {J_n^f}^T + R_n$$

Filtering step (Measurement update):

$$K_n = \bar{\Sigma}_n J_n^{h^T} \left(J_n^h \bar{\Sigma}_n J_n^{h^T} + Q_n \right)^{-1}$$

$$z_n = \bar{z}_n + K_n (y_n - h(\bar{z}_n))$$

$$\Sigma_n = \left(1 - K_n J_n^h\right) \bar{\Sigma}_n$$





State Space Models (SSMs) and Kalman Filtering (KF)

Unscented Kalman Filter (UKF)



Motivations

There are two cases in which both KF and EKF perform poorly:

1. When the covariance is large.
2. When the transition and/or measurement functions are highly non-linear.

To overcome these limitations, we can use Unscented Kalman Filter which is based on the concept of sigma points.

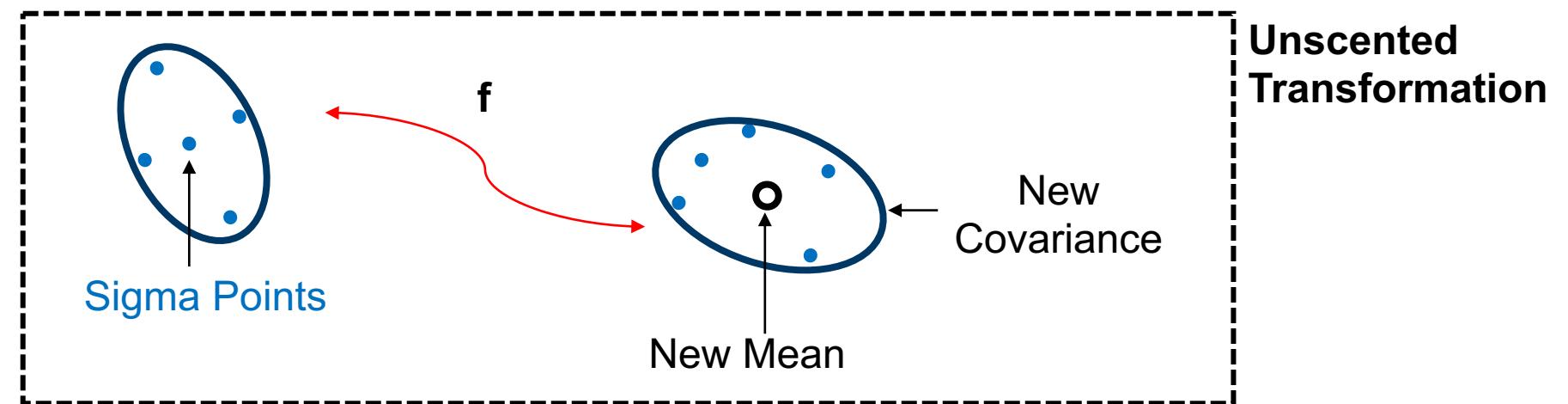
Unscented Kalman Filter (UKF): the basic idea

The Unscented Kalman Filter makes use of the deterministic sampling technique, namely the **unscented transformation**

→ Pick up minimal set of sigma points

Then, sigma points are propagated through a non-linear function f

→ We obtain new mean and covariance estimates



Sigma points

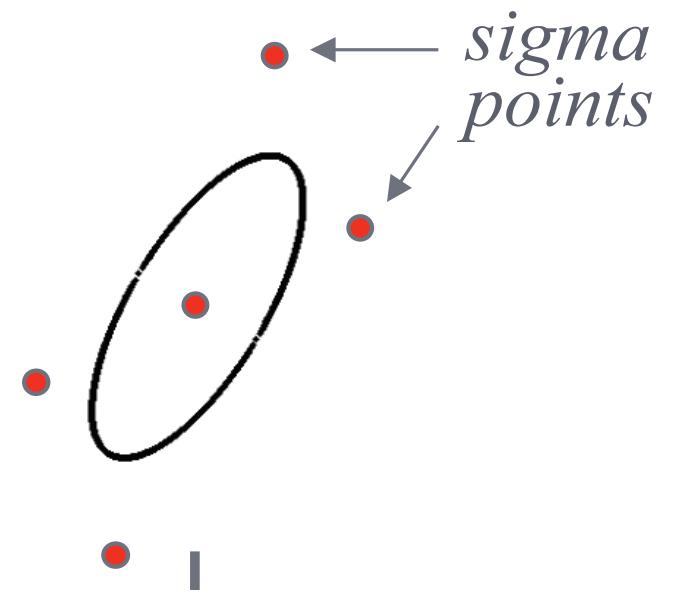
Let's call sigma points a set of weighted points $\{z_i\}_{i=0}^L$ chosen deterministically.

We assume these points capture the mean and covariance of the random variable z , i.e.,

$$\mu \approx \sum_{l=0}^L w_l \mathbf{z}_l,$$

$$\Sigma \approx \sum_{l=0}^L w_l (\mathbf{z}_l - \mu_n)(\mathbf{z}_l - \mu_n)'$$

where $\{w\}_{i=0}^L$ is a set of weights, with $\sum_i w_i = 1$



Compared to the EKF, we do not approximate a non-linear function but we estimate a Gaussian distribution.

Sigma points

Let μ and Σ be the mean and the covariance of \mathbf{z} .

The $2D + 1$ sigma points and weights are defined as follows:

$$\begin{aligned}\mathbf{z}_0 &= \mu, & w_0 &= \frac{\kappa}{D + \kappa}, & l &= 0, \\ \mathbf{z}_l &= \mu + \left[\sqrt{(D + \kappa)\Sigma} \right]_l, & w_l &= \frac{1}{2(D + \kappa)}, & l &= 1, \dots, D, \\ \mathbf{z}_{l'} &= \mu - \left[\sqrt{(D + \kappa)\Sigma} \right]_{l'}, & w_{l'} &= \frac{1}{2(D + \kappa)}, & l' &= D + 1, \dots, 2D,\end{aligned}$$

where κ is a scale parameter (determining the radius of the sigma points from the mean).

- The sigma points capture the mean and covariance of \mathbf{z} .
- When propagated through any nonlinear system, the transformed sigma points capture the predictive and filtering mean and covariance.

Unscented Kalman Filter (UKF)

The Unscented Kalman Filter is simply two applications of the unscented transformation,

- one to compute the predictive density, i.e., $p(z_n|y_{1:n-1})$
- and another to compute the filtering density, i.e., $p(z_n|y_{1:n})$.

Unscented Kalman Filter (UKF)

The Unscented Kalman Filter is simply two applications of the unscented transformation,

- one to compute the predictive density, i.e., $p(z_n|y_{1:n-1})$
- and another to compute the filtering density, i.e., $p(z_n|y_{1:n})$.

In the first step, the old state $\mathcal{N}(\mu_{n-1}, \Sigma_{n-1})$ is passed through the transition function f in order to approximate the predictive density $\mathcal{N}(\bar{\mu}_n, \bar{\Sigma}_n)$.

Let $z_{n-1}^0 = \{z_i\}_{i=0}^L$ be a set of sigma points; we pass them through the function f and obtain:

$$z_{n-1}^{*i} = f(z_{n-1}^{0i})$$

and the mean and covariance of the new points are:

$$\bar{\mu}_n = \sum_{i=0}^{2D} w_i z_n^{*i} \quad \bar{\Sigma}_n = \sum_{i=0}^{2D} w_i (z_n^{*i} - \bar{\mu}_n)(z_n^{*i} - \bar{\mu}_n)^T + \mathbf{R}_n$$

Unscented Kalman Filter (UKF)

In the second step, we approximate the likelihood $p(y_n, | z_n)$ by passing the predictive density $\mathcal{N}(\bar{\mu}_n, \bar{\Sigma}_n)$ through the observation function h .

Passing the sigma points through the function h we obtain:

$$\bar{y}_n^{*i} = h(z_n^{0i})$$

Again we compute mean and covariance:

$$\hat{y}_n = \sum_{i=0}^{2D} w_i \bar{y}_n^{*i}$$

$$S_n = \sum_{i=0}^{2D} w_i (\bar{y}_n^{*i} - \hat{y}_n)(\bar{y}_n^{*i} - \hat{y}_n)^T + Q_n$$

Unscented Kalman Filter (UKF)

Finally, we can use the Bayes rule for Gaussian to get the filtering density (or posterior) $p(z_n|y_{1:n})$.

We use the following formulas to compute the covariance between z and y

$$\bar{\Sigma}_n^{z,y} = \sum_{i=0}^{2D} w_i (\bar{z}_n^{*i} - \bar{\mu}_n)(\bar{y}_n^{*i} - \hat{y}_n)^T$$

the Kalman gain

$$\mathbf{K}_n = \bar{\Sigma}_n^{z,y} \mathbf{S}_n^{-1}$$

And estimating mean and covariance of the filtering density

$$\mu_n = \bar{\mu}_n + \mathbf{K}_n(\mathbf{y} - \hat{\mathbf{y}}) \quad \Sigma_n = \bar{\Sigma}_n - \mathbf{K}_n \mathbf{S}_n \mathbf{K}_n^T$$

UKF: An algorithmic view

The simplest choice for sigma points:

$$\{s^0, \dots, s^{2D}\}_{n-1}$$

$$s_{n-1}^0 = z_{n-1}$$

$$s_{n-1}^i = z_{n-1} + \sqrt{\frac{D}{1-w_0}} A_i, i = 1, \dots, D$$

$$s_{n-1}^{D+i} = z_{n-1} - \sqrt{\frac{D}{1-w_0}} A_i, i = 1, \dots, D$$

$$w_i = \frac{1-w_0}{2D}, i = 1, \dots, 2D$$

$$AA^T = \Sigma_{n-1}$$

UKF: An algorithmic view

Prediction step (time update):

$$\{s^0, \dots, s^{2D}\}_{n-1} \quad \bar{z}_n = \sum_{i=0}^{2D} w_i f(s_{n-1}^i) \quad \bar{\Sigma}_n = \sum_{i=0}^{2D} w_i (f(s_{n-1}^i) - \bar{z}_n) (f(s_{n-1}^i) - \bar{z}_n)^T + R_n$$

Filtering step (Measurement update):

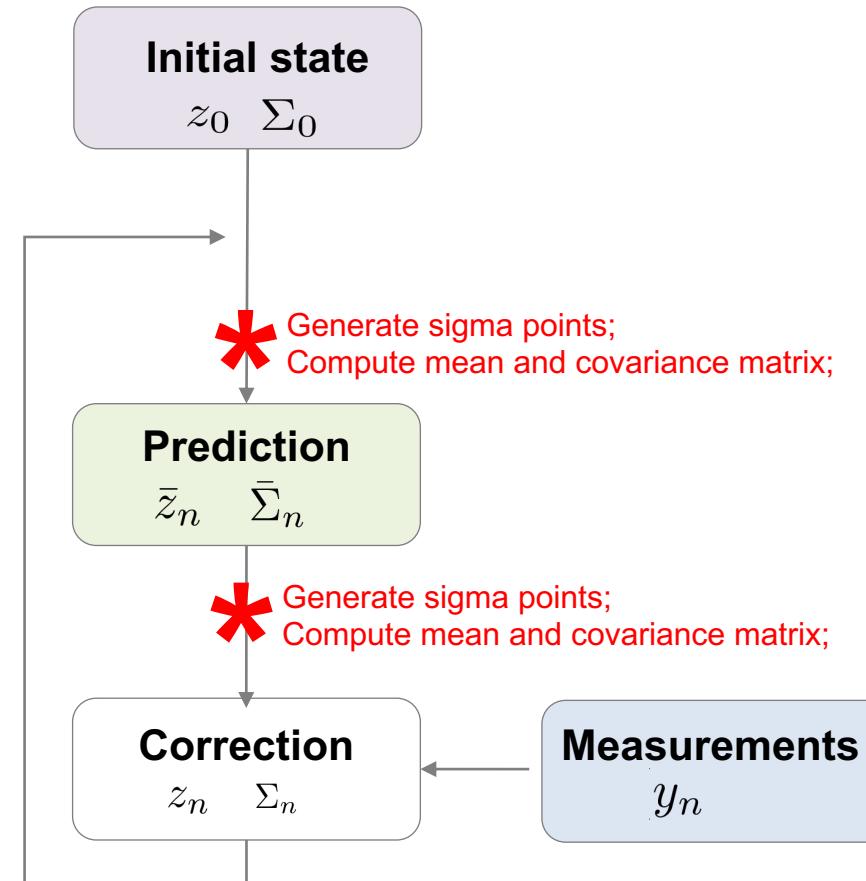
$$\{\bar{s}^0, \dots, \bar{s}^{2D}\}_{n-1} \quad \bar{y}_n = \sum_{i=0}^{2D} w_i h(\bar{s}_{n-1}^i) \quad \bar{S}_n = \sum_{i=0}^{2D} w_i (h(\bar{s}_{n-1}^i) - \bar{y}_n) (h(\bar{s}_{n-1}^i) - \bar{y}_n)^T + Q_n$$

$$\bar{\Sigma}_n^{z,y} = \sum_{i=0}^{2D} w_i (\bar{s}_{n-1}^i - \bar{z}_n) (h(\bar{s}^i) - \bar{y}_n)^T$$

$$K_n = \bar{\Sigma}_n^{z,y} \bar{S}_n^{-1}$$

$$z_n = \bar{z}_n + K_n (y_n - \bar{y}_n) \quad \Sigma_n = \bar{\Sigma}_n - K_n \bar{S}_n K_n^T$$

UKF: An algorithmic view





State Space Models (SSMs) and Kalman Filtering (KF)

Recap



- State space models
- Kalman Filtering
- Extended Kalman Filter
- Unscented Kalman Filter

Recap

Critical comparison

Estimator	State-transition / Measurement models assumptions	Assumed noise distribution	Computational cost
Kalman Filter	Linear	Gaussian	Low
Extended Kalman Filter	Non-linear (but locally linear)	Gaussian	Low / Medium (depending on the difficulty of computing the Jacobian)
Unscented Kalman Filter	Non-linear	Gaussian	Medium



Machine Learning for Time Series

(MLTS or MLTS-Deluxe Lectures)

Dr. Dario Zanca

Machine Learning and Data Analytics (MaD) Lab
Friedrich-Alexander-Universität Erlangen-Nürnberg
25.10.2022

-
- Time series fundamentals and definitions (2 lectures)
 - Bayesian Inference (1 lecture)
 - Gaussian processes (2 lectures)
 - State space models (2 lectures) ←
 - Autoregressive models (1 lecture)
 - Data mining on time series (1 lecture)
 - Deep learning on time series (4 lectures)
 - Domain adaptation (1 lecture)

Review concept: State Space Models (SSMs)

SSMs are characterized by:

- Transition density: $p(z_n|z_{n-1})$
- Observation density: $p(y_n|z_n)$

Joint density can be expressed using the *chain rule*:

$$p(z_{1:n}|y_{1:n}) = p(z_1) \prod_{i=2}^n p(z_i|z_{i-1}) \prod_{i=1}^n p(y_i|z_i)$$

where $z_{1:n} = (z_1, \dots, z_n)$ and $y_{1:n} = (y_1, \dots, y_n)$.

Filtering is the task of estimating $p(z_n|y_{1:n})$.

Review concept: Bayesian filtering

Let $p(z_{n-1}|y_{1:n-1})$ be the filtering density at time step $n - 1$ and we wish to determine $p(z_n|y_{1:n})$.

We can use the following iterative steps:

- Prediction step:

$$p(z_n|y_{1:n-1}) = \int p(z_n|z_{n-1}) p(z_{n-1}|y_{1:n-1}) dz_n$$

- Correction step:

$$p(z_n|y_{1:n}) = \frac{p(y_n|z_n) p(z_n|y_{1:n-1})}{p(y_n|y_{1:n-1})}$$

If the variables are normally distributed and the transitions are linear, the Bayes filter becomes equal to the Kalman filter.

In this lecture...

- 1. Monte Carlo methods**
- 2. Particle filtering: theory**
- 3. Particle filtering: example and algorithmic view**



Sequential Monte Carlo and Particle Filtering

Monte Carlo methods



Motivation

The fundamental goal of this section is “**how to find the expectation of a function $f(z)$ with respect to a probability distribution $p(z)$** ”.

$$\mathbb{E}[f] = \int f(z)p(z)dz$$

Motivation

The fundamental goal of this section is “**how to find the expectation of a function $f(z)$ with respect to a probability distribution $p(z)$** ”.

$$\mathbb{E}[f] = \int f(z)p(z)dz$$

Sampling methods can be used to approximate this expectation using a set of samples $z^s \sim p(z)$, by the finite sum:

$$\frac{1}{S} \sum_{s=1}^S f(z^s)$$

where $S \in \mathbb{Z}^+$.

Motivation

However, possible problems are:

- Samples z^S might not be independent
- Depending on f , expectation might be dominated by examples with small probability
→ We need relatively large sample size S .

Monte Carlo methods (MC)

Monte Carlo methods include a wide class of **algorithms that rely on random sampling** to obtain numerical results.

- Use randomness to solve problem that might be deterministic in principle.
- Used, e.g., in optimization, numerical integration and for generating draws from a probability distribution.

General idea of MC methods:

- Define a domain of possible inputs
- Generate input randomly from a probability distribution
- Perform deterministic computations on the inputs
- Aggregate results

Rejection sampling

Rejection sampling is a Monte Carlo algorithm to sample data from a sophisticated (“difficult to sample from”) distribution with the help of a proxy distribution.

Let us suppose that:

- We wish to sample from a “non-standard” distribution $p(z)$
- Sampling directly from $p(z)$ is difficult
- We are able to evaluate $p(z)$ for any z , up to a certain (unknown) normalizing constant Z_p :

$$p(z) = \tilde{p}(z)/Z_p$$

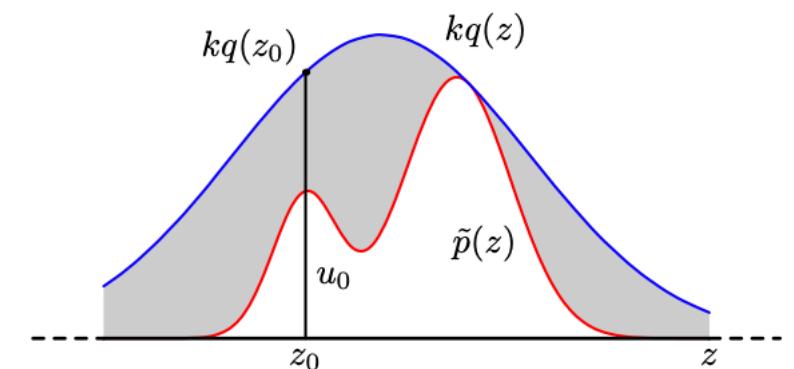
To apply rejection sampling, we make use of a simpler distribution $q(z)$, also called **proposal distribution** → We are able to readily draw samples from $q(z)$

Rejection sampling

Samples are generated from the proposal distribution $q(z)$ and rejected if they fall between the unnormalized $\tilde{p}(z)$ and the scaled distribution $kq(z)$.

→ In the side figure, samples are rejected if they fall in the grey area.

→ Samples are accepted with probability $\frac{\tilde{p}(z)}{kq(z)}$

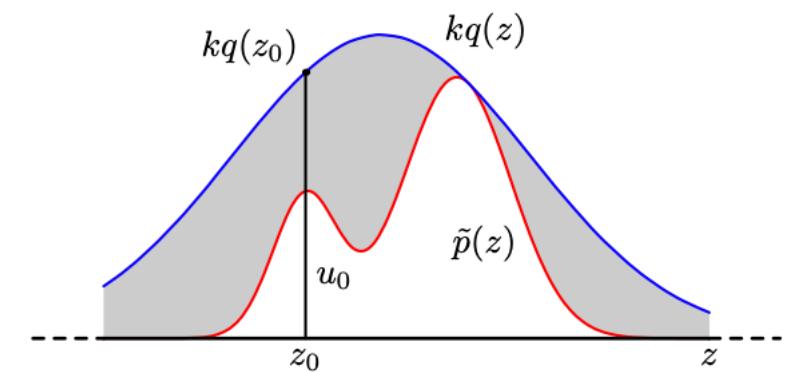


Note: k is a constant value which is chosen such that $kq(z) \geq \tilde{p}(z)$ for all values of z .

Rejection method

The success of the rejection method depend on the success in determining a suitable value for the constant k .

→ This is impractical in many cases, which leads to very small acceptance rates.



Importance sampling

Importance sampling is a method which allows approximating expectations directly.

Suppose, similarly to the previous case, that:

- It is impractical to draw samples from $p(z)$
- But, we can evaluate $p(z)$ easily for any z

Uniform sampling from the space of z is inefficient (high-dimensionality) and only few samples will have significant contribution.

→ We would like to chose samples from regions where $p(z)$ is large.

Importance sampling

We draw samples from a proposal distribution $q(z)$. Then,

$$\mathbb{E}[f] = \int f(z)p(z)dz = \int f(z)\frac{p(z)}{q(z)}q(z)dz \approx \frac{1}{S}\sum_{s=1}^S \frac{p(\mathbf{z}^s)}{q(\mathbf{z}^s)}f(\mathbf{z}^s)$$

The quantity $\frac{p(z^s)}{q(z^s)}$ measures the importance of each sample and are called **importance weights**.

- Compared to rejection sampling, no samples are rejected.
- Importance sampling do not provide itself a mechanism for drawing samples from a distribution $q(z)$.

Sampling-importance-resampling approach

The **sampling-importance-resampling** approach also makes use of a proposal distribution $q(z)$ and **avoids determining a constant k .**

It consists of three *general* steps:

1. Sampling of $\{z_1, \dots, z_s\}$ from the proposal distribution $q(z)$
2. Construction of importance weights $\{w_1, \dots, w_s\}$
3. Resampling from a discrete distribution with probabilities given by the weights

Property. The resulting samples are approximately distributed according to $p(z)$ and the distribution becomes correct for $S \rightarrow \infty$.



Sequential Monte Carlo and Particle Filtering

Particle Filtering (PF): Theory



Motivations

Estimator	State-transition / Measurement models assumptions	Assumed noise distribution	Computational cost
Kalman Filter	Linear	Gaussian	Low
Extended Kalman Filter	Non-linear (but locally linear)	Gaussian	Low / Medium (depending on the difficulty of computing the Jacobian)
Unscented Kalman Filter	Non-linear	Gaussian	Medium

Particle Filtering (PF)

We can use sampling-importance-resampling formalism to obtain a **sequential Monte Carlo**, also said **particle filtering**.

PF is a Monte Carlo (or simulation-based) approach for recursive Bayesian inference.

- It approximates the prediction-correction cycle
- It can be **used for not linear-Gaussian systems** to make tractable inference algorithms
- It is widely applied in many areas (e.g., tracking, forecasting, online parameter learning, ...)

 The term “particle filters” originated in reference to mean-field interacting particle methods used in fluid mechanics since early 1960s (Del Moral, 1966).

Particle Filtering (PF)

First, we update the belief state using importance sampling.

If the proposal distribution has the form $q(z_{1:n}^s | y_{1:n})$, then the importance weights are given by

$$w_n^s \propto \frac{p(z_{1:n}^s | y_{1:n})}{q(z_{1:n}^s | y_{1:n})}$$

which can be normalized as follow:

$$\hat{w}_n^s \propto \frac{w_n^s}{\sum_{s'} w_n^{s'}}$$

Particle Filtering (PF)

We observe that we can rewrite the numerator recursively as follows:

$$\begin{aligned} p(z_{1:n}|y_{1:n}) &= \frac{p(y_n|z_{1:n}, y_{1:n-1})p(z_{1:n}|y_{1:n-1})}{p(y_n|y_{1:t-1})} \\ &= \frac{p(y_n|z_n)p(z_n|z_{1:n-1}, y_{1:n-1})p(z_{1:n-1}|y_{1:n-1})}{p(y_n|y_{1:t-1})} \\ &\propto p(y_n|z_n)p(z_n|z_{n-1})p(z_{1:n-1}|y_{1:n-1}) \end{aligned}$$



Markov property

And, similarly, the denominator:

$$q(z_{1:n}|y_{1:n}) = q(z_n|z_{1:n-1}, y_{1:n}) q(z_{1:n-1}|y_{1:n-1})$$

Particle Filtering (PF)

Therefore, we use this formulation to derive an iterative update for the weights:

$$\begin{aligned} w_n^s &\propto \frac{p(z_{1:n}^s | y_{1:n})}{q(z_{1:n}^s | y_{1:n})} \\ &\propto \frac{p(y_n | z_n^s) p(z_n^s | z_{n-1}^s) p(z_{1:n-1}^s | y_{1:n-1})}{q(z_n^s | z_{1:n-1}^s, y_{1:n}) q(z_{1:n-1}^s | y_{1:n-1})} \\ &= w_{n-1}^s \frac{p(y_n | z_n^s) p(z_n^s | z_{n-1}^s)}{q(z_n^s | z_{1:n-1}^s, y_{1:n})} \end{aligned}$$

Hence, we can approximate the posterior filtered density using

$$p(z_n | y_{1:n}) \approx \sum_{s=1}^S \hat{w}_n^s \delta_{z_n^s}(z_n)$$

Particle Filtering: the degeneracy problem

The basic sequential importance fails after a few steps because most of the particles will have negligible weights.

→ This problem is known as **degeneracy problem** and it occurs when sampling in high dimensional spaces

We can quantify the degree of degeneracy by using the effective sampling size, defined by:

$$S_{eff} = \frac{1}{\sum_{s=1}^S (w_n^s)^2}$$

When the variance of the weights is too large, we are wasting resources updating particles with negligible weights.

Particle filtering: the resampling step

The degeneracy problem can be solved by **adding a resampling step**.

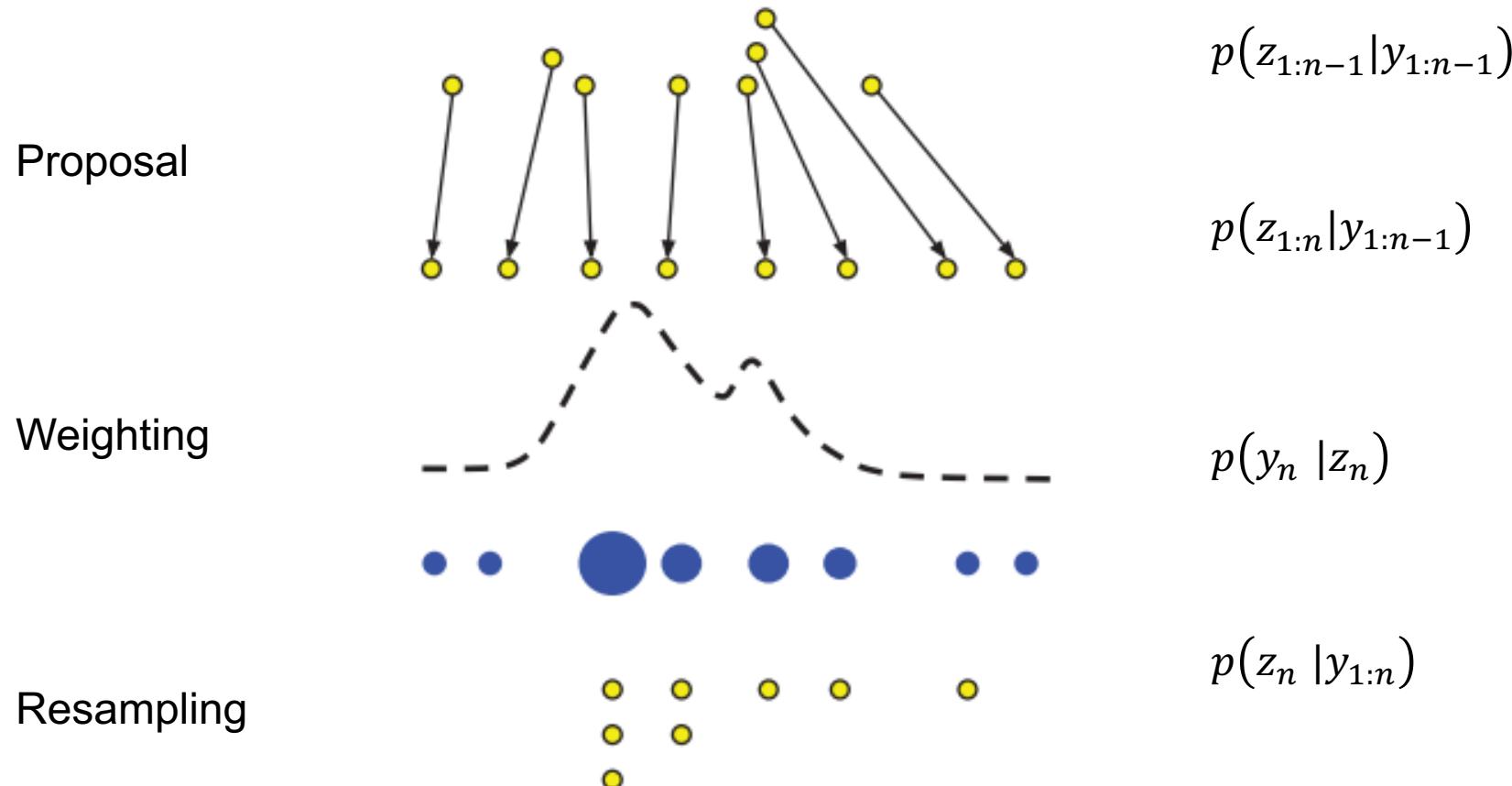
Wheneven the effective sampling size S_{eff} drops below a threshold:

- we eliminate particles with low weights and
- we create replicates of the survival particles.

In particular, we generate a new set $\{z_n^{s*}\}_{s=1}^S$ by sampling replacement S times according to the weighted distribution obtained previously $\sum_{s=1}^S \hat{w}_n^s \delta_{z_n^s}(z_n)$.

The result is an *i.i.d. unweighted sample* from the discrete density, so we set the new weights to $w_n^s = 1/S$.

PF: the overall scheme



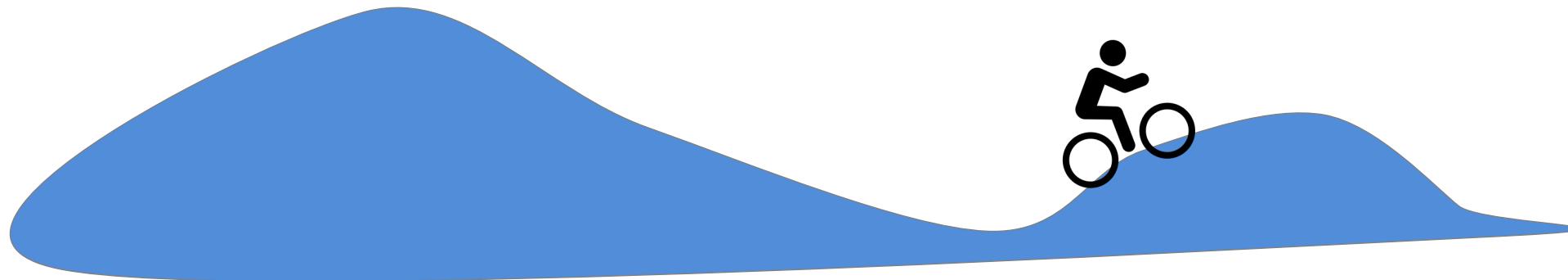


Sequential Monte Carlo and Particle Filtering

Particle Filtering (PF): Example and Algorithmic View

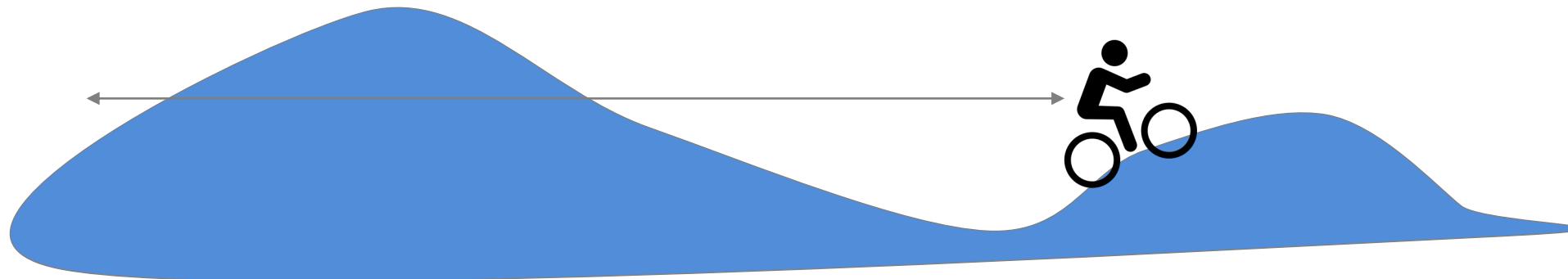


PF: Example



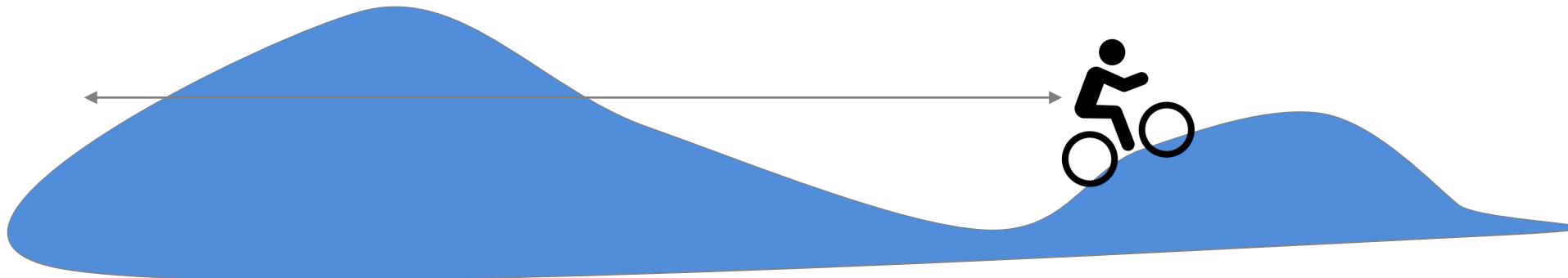
PF: Example

1. We want to estimate the **horizontal position** of the cyclist.



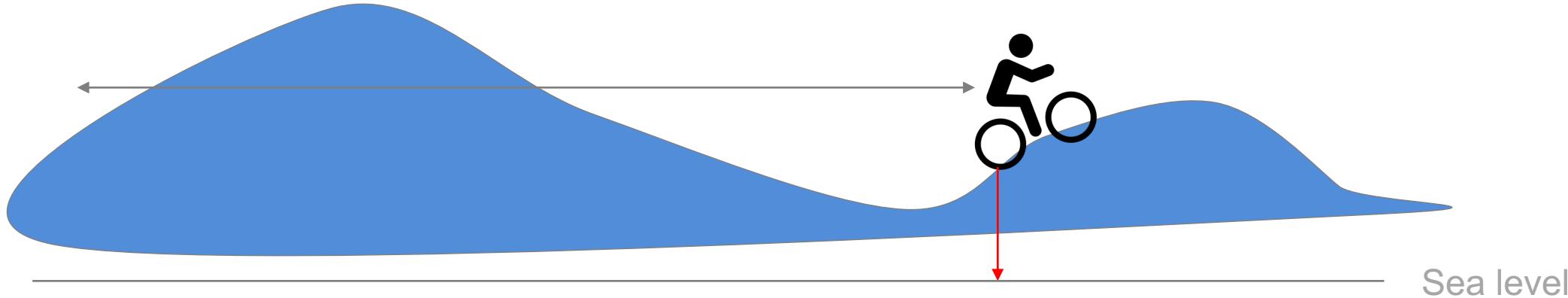
PF: Example

1. We want to estimate the **horizontal position** of the cyclist.
2. We have **knowledge** about the hills' morphology

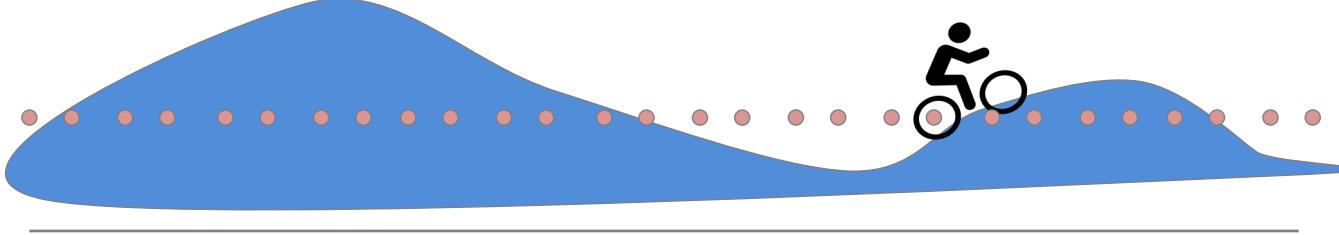


PF: Example

1. We want to estimate the **horizontal position** of the cyclist.
2. We have **knowledge** about the hills' morphology
3. We receive noisy **measures of the altitude** (e.g., in relation to sea level).



PF: Example



Particles (time n)

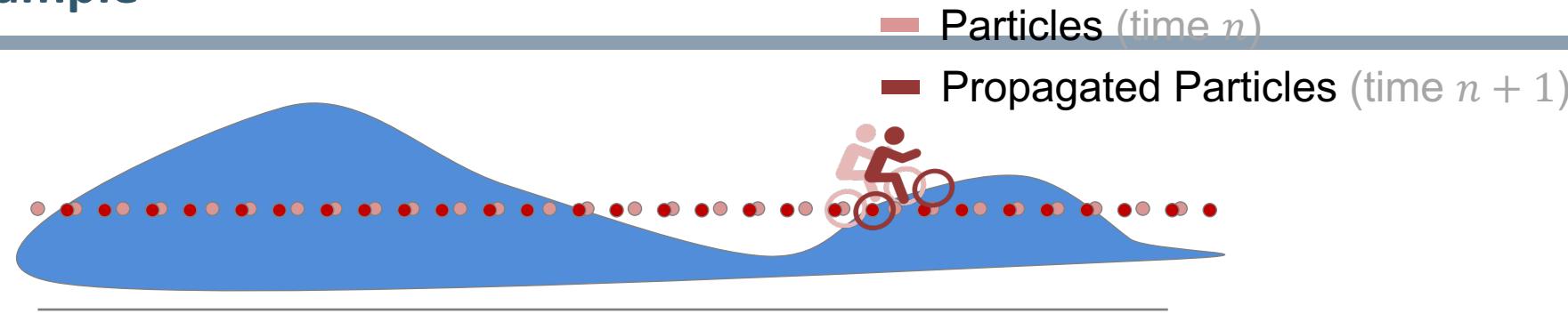
We draw a **set of particles** $\langle z_n^s, w_n^s \rangle_{s \in \{1, \dots, S\}}$

- n is the time index,
- z_n^s is a state hypothesis,
- w_n^s corresponding weights.

These samples represent the prior probability:

$$p(z_n | y_1, \dots, y_n) \simeq \sum_{s=1}^S w_n^s * \delta(z_n^s)$$

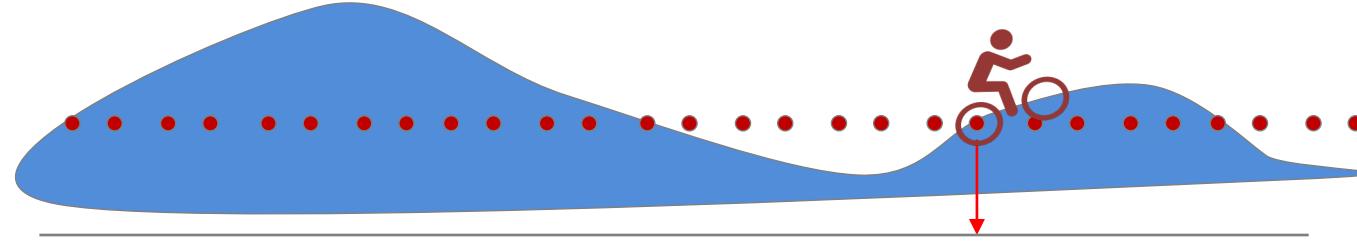
PF: Example



In the **prediction step**, we use our transition model f to **propagate** particles forward in time:

$$z_{n+1}^s = f(z_n^s) + \epsilon$$

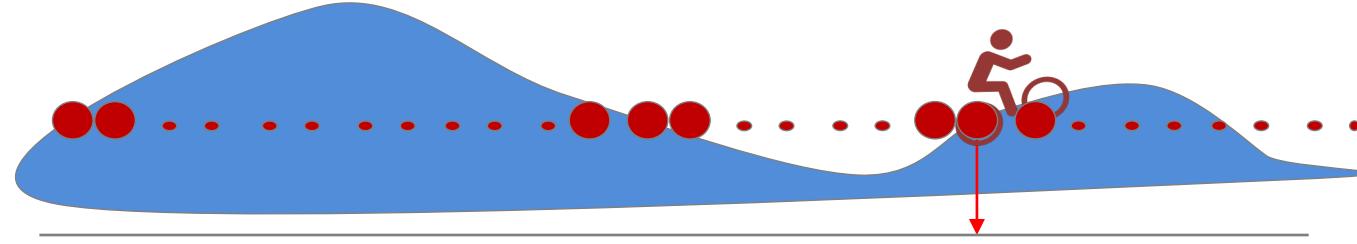
PF: Example



In the **correction step**, we **compute particle weights** based on the new sensory information y_{n+1} :

$$w_{n+1}^s = w_n^s * p(y_{n+1} | z_{n+1}^s)$$

PF: Example



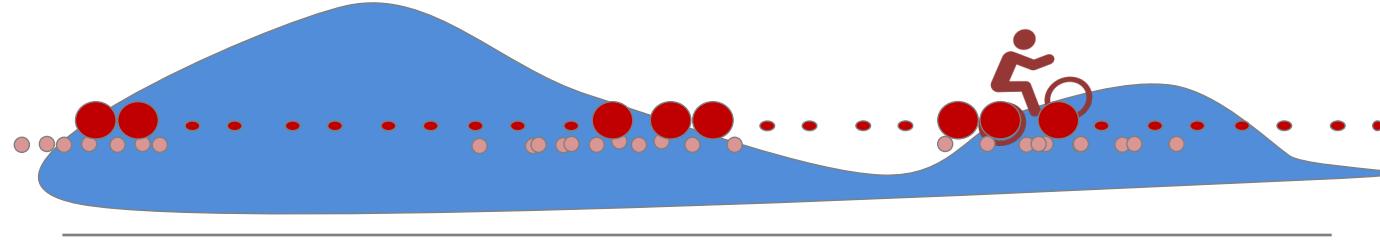
In the **correction step**, we **compute particle weights** based on the new sensory information y_{n+1} :

$$w_{n+1}^s = w_n^s * p(y_{n+1} | z_{n+1}^s)$$

Now the distribution $p(z_{n+1} | y_{1:t+1})$ is represented by the particle set:

$$\langle z_{n+1}^s, w_{n+1}^s \rangle_{s \in \{1, \dots, S\}}$$

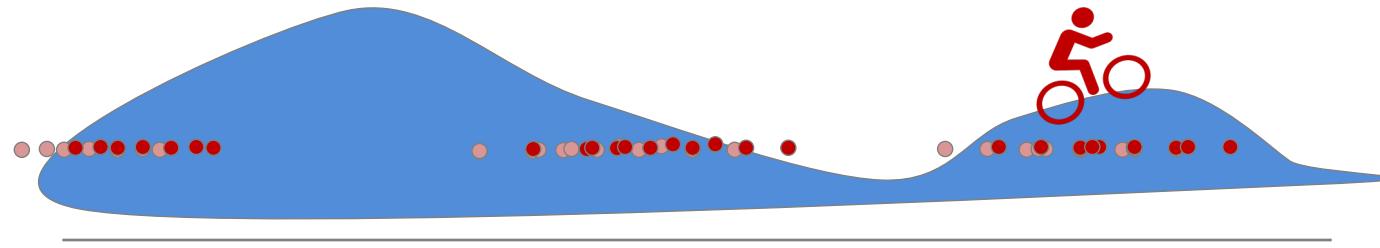
PF: Example



We **repeat** the previous steps:

- Sampling again from the new distribution (we have more particles close to the more likely state's hypothesis).

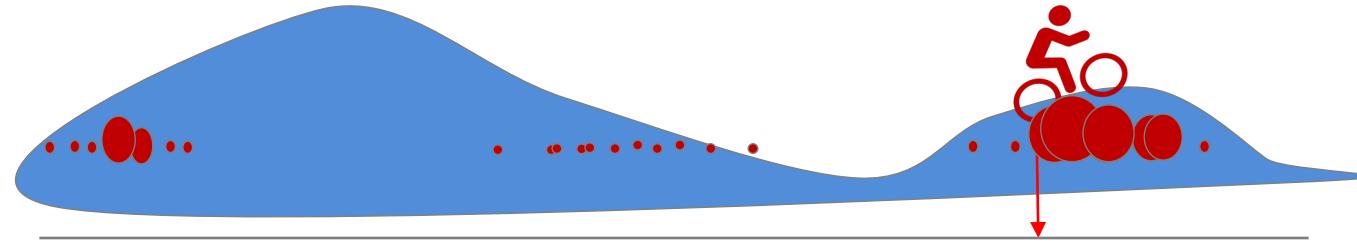
PF: Example



We **repeat** the previous steps:

- Sampling again from the new distribution (we have more particles close to the more likely state's hypothesis).
- We propagate again through time.

PF: Example



We **repeat** the previous steps:

- Sampling again from the new distribution (we have more particles close to the more likely state's hypothesis).
- We propagate again through time.
- We perform again a correction, based on the new measurement, and update the particles' weights.
- ...

PF: An algorithmic view

Initially sample S particles z_1^S

For $n = 1, \dots, N$

 For $s = 1, \dots, S$

 Draw $z_n^s \sim q(z_n^s | z_{n-1}^s, y_n)$

 update and normalize w_{n+1}^s

 if $S_{eff} < S_{min}$:

 Resample particles

 Re-initialize weights

 End

End

End



Sequential Monte Carlo and Particle Filtering

Recap



Recap

- Monte Carlo methods
 - Rejection method
 - Importance sampling
 - Sampling-importance-resampling approach
- Particle filtering

Particle Filtering: pros and cons

Advantages:

- Ability to represent arbitrary densities
- Adaptive focusing on probable regions of state-space
- Dealing with non-Gaussian noise

Disadvantages:

- High computational complexity
- It is difficult to determine optimal number of particles
- Number of particles increase with increasing model dimension
- Potential problems: degeneracy



Machine Learning for Time Series

(MLTS or MLTS-Deluxe Lectures)

Dr. Dario Zanca

Machine Learning and Data Analytics (MaD) Lab
Friedrich-Alexander-Universität Erlangen-Nürnberg
20.12.2022

-
- Time series fundamentals and definitions (2 lectures)
 - Bayesian Inference (1 lecture)
 - Gaussian processes (2 lectures)
 - State space models (2 lectures)
 - Autoregressive models (1 lecture) ←
 - Data mining on time series (1 lecture)
 - Deep learning on time series (4 lectures)
 - Domain adaptation (1 lecture)

Review concept: Stochastic process

Non-deterministic time series can be regarded as manifestations (equiv., realization) of a **stochastic process**, which is defined as a set of random variables $\{X_t\}_{t \in \{1, \dots, T\}}$

Even if we were to imagine having observed the process for an infinite period T of time, the infinite sequence

$$S = \{\dots, s_{t-1}, s_t, s_{t+1}, \dots\} = \{s_t\}_{t=-\infty}^{+\infty}$$

would still be a single **realization** from that process.

Still, if we had a battery of N computers generating series $S^{(1)}, \dots, S^{(N)}$, and considering selecting the observation at time t from each series,

$$\{s_t^{(1)}, \dots, s_t^{(N)}\}$$

this would be described as a sample of N realizations of the random variable X_t

Review concept: Autocovariance

Given any particular realization $S^{(i)}$ of a stochastic process (i.e., a time series), we can define the vector of the $j + 1$ most recent observations

$$x_t^i = [s_{t-j}^{(i)}, \dots, s_t^{(i)}]$$

We want to know the probability distribution of this vector x_t^i across realizations. We can calculate the **j -th autocovariance**

$$\gamma_{jt} = E(X_t - \mu_t)(X_{t-j} - \mu_{t-j})$$

Review concept: Autocorrelation function (ACF)

We can express the linear predictability of X_t from an adjacent value X_s , using the **autocorrelation function**:

$$\rho(s, t) = \frac{\gamma_{st}}{\sqrt{\gamma_{ss}\gamma_{tt}}}$$

where γ is the autocovariance defined previously.

Review concept: Stationarity

There are two types of stationarity.

A process is said **strictly stationary** if the joint distribution of $X_{t_1:t_2}$ is the same as that of $X_{t_1+h:t_2+h}$.

The term h is called **lag**.

For strictly stationary time series, all statistics do not depend on time.

A process is said **weakly stationary** if it has:

- $\mu = \text{const.}$
- $\sigma^2 < \infty$
- $\gamma_{jt} = \gamma_{j+h,t+h}$

A weakly stationary time series has finite variation, constant first moment, and that the second moment only depends on $h = t - j$.

Review concept: Partial autocorrelation function (PACF)

For stationary time series, the **partial autocorrelation function** expresses the correlation between X_t and an adjacent value X_s , but “removes” the effect of all values in between:

$$\phi_{11} = \text{corr}(X_{t+1}, X_t) = \rho_1$$

$$\phi_{hh} = \text{corr}\left(X_{t+h} - P_{t,h}(X_{t+h}), X_t - P_{t,h}(X_t)\right) = \rho_h$$

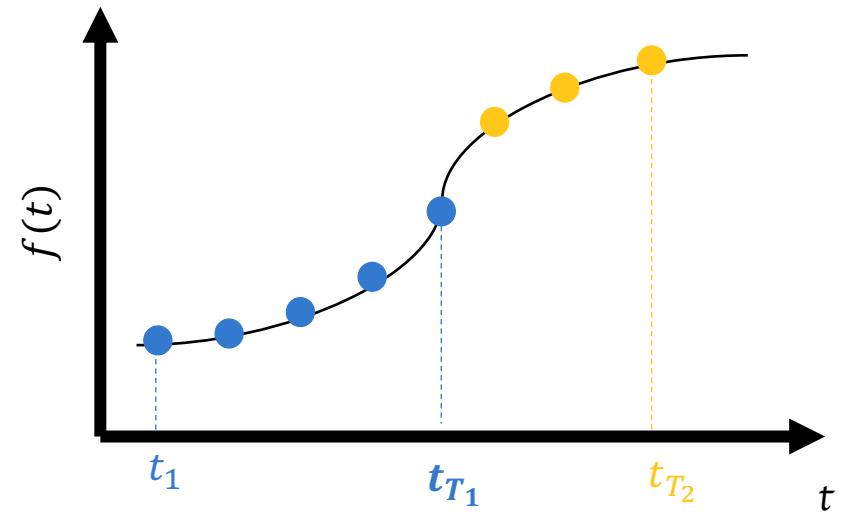
for $h \geq 2$, where $P_{t,h}$ is the surjective operator of orthogonal projection onto the linear subspace spanned by the intermediate values $X_{t+1}, \dots, X_{t+h-1}$.

Review concept: Time series forecasting

Let $S = \{s_1, \dots, s_{T_1}, s_{T_1+1}, \dots, s_{T_2}\}$ be a time series, with s_i being the i -th observation collected at time t_i , and $t_i < t_j, \forall j$.

Then, a time series forecasting task is about predicting future values of a time series given some past data, i.e.,

$$f(s_1, \dots, s_{T_1}) = (s_{T_1+1}, \dots, s_{T_2})$$



In this lecture...

- Linear processes
 - Autoregressive processes (AR)
 - Moving average processes (MA)
- Combining AR and MA:
 - ARMA
 - ARIMA



Autoregressive models

AR and MA models



White noise

The basic building block for all the processes considered in this lecture is the **white noise**, defined as a sequence

$$\{e_t\}_{t=-\infty}^{+\infty}$$

whose elements have zero mean and variance σ^2 , and are *uncorrelated* across time, i.e.,

- $\mathbb{E}(e_t) = 0$ (zero mean)
- $\mathbb{E}(e_t^2) = \sigma^2$ (variance)
- $\mathbb{E}(e_t e_\tau) = 0$ (zero autocovariance, i.e., uncorrelated)

If $e_t \sim \mathcal{N}(0, \sigma^2)$, then we have a so-called **Gaussian white noise**.

Random walk

A random walk is a stochastic random process that describes a path started at y_0 and consisting of random steps.

$$y_t = y_{t-1} + e_t$$

Equiv., for $t \geq 1$:

$$y_t = y_0 + \sum_{i=1}^t e_i$$

where e_i can be regarded as random variables of a white noise process.

Linear process representation

A linear process can be represented as an **infinite moving average process**, starting from a white noise $\{e_t\}_{t=-\infty}^{+\infty}$, as

$$\begin{aligned} y_t &= \mu + e_t + \psi_1 e_{t-1} + \psi_2 e_{t-2} + \dots \\ &= \mu + e_t + \sum_{j=1}^{+\infty} \psi_j e_{t-j} \\ &= \mu + \Psi(q^{-1}) e_t \end{aligned}$$

where,

- ψ_i are constant values
- q^{-m} is the *backshift operator*, such that $q^{-m} e_t = e_{t-m}$
- $\Psi(q^{-1}) = 1 + \psi_1 q^{-1} + \psi_2 q^{-2} + \dots = \sum_{j=0}^{+\infty} \psi_j q^{-j}$
is a **linear filter**.

If the sequence ψ_1, ψ_2, \dots , has finite sum $\sum_i^{\infty} \psi_i < \infty$, then the filter is stable and the process y_t is stationary.

Linear process representation

Alternatively, a linear process can be represented with respect to its previous values as an **infinite autoregressive process**:

$$y_t = \mu + e_t + \pi_1 y_{t-1} + \pi_2 y_{t-2} + \dots$$

$$y_t = \mu + e_t + \sum_{j=1}^{+\infty} \pi_j y_{t-j}$$

$$\Pi(q^{-1})y_t = \mu + e_t$$

where, similarly,

- π_i are constant values
- q^{-m} is the *backshift operator*, such that $q^{-m}e_t = e_{t-m}$
- $\Pi(q^{-1}) = 1 + \pi_1 q^{-1} + \pi_2 q^{-2} + \dots = \sum_{j=0}^{+\infty} \pi_j q^{-j}$ is a **linear filter**.

Linear process representation

The previous two formulations are algebraically equivalent, in fact:

$$y_t = \mu + e_t + \sum_{j=1}^{+\infty} \pi_j y_{t-j} \quad (\text{infinite autoregressive process})$$

$$y_t = \mu + e_t + \pi_1 q^{-1} y_t + \dots$$

$$y_t - \pi_1 q^{-1} y_t - \dots = \mu + e_t$$

$$(1 - \pi_1 q^{-1} - \dots) y_t = \mu + e_t$$

$$\Pi(q^{-1}) y_t = \mu + e_t$$

If the linear filter $\Pi(q^{-1})$ is **invertible**, then:

$$y_t = \bar{\mu} + \frac{1}{\Pi(q^{-1})} e_t \quad (\text{infinite moving average process})$$

Autoregressive models (AR)

Autoregressive models are based on the idea that the value of a time series at time t can be expressed as a linear combination of n past values, up to a random error:

$$AR(n): y_t = a_1 y_{t-1} + a_2 y_{t-2} + \cdots + a_n y_{t-n} + e_t$$

Where:

- n is the model's order
- a_1, \dots, a_n are the model's parameters, $a_n \neq 0$

In other words, the hyper-parameter n represents how far back to look for dependences with previous values in the time series.

Autoregressive models (AR)

We can simplify the notation for $AR(n)$ using the backshift operator:

$$y_t = a_1 y_{t-1} + a_2 y_{t-2} + \cdots + a_n y_{t-n} + e_t$$

$$y_t - a_1 y_{t-1} - \cdots - a_n y_{t-n} = e_t$$

$$(1 - a_1 q^{-1} - \cdots - a_n q^{-n}) y_t = e_t$$

$$A(q^{-1}) y_t = e_t$$

where $A(q^{-1})$ is called **autoregressive operator**.

Example: AR(0)

The simplest autoregressive model is **AR(0)**, which has no dependences between values in the time series.

$$AR(0): y_t = e_t$$

→ AR(0) is equivalent to a white noise process.

Example: AR(1)

The first order autoregressive model AR(1) can be written as:

$$AR(1): y_t = a_1 y_{t-1} + e_t$$

Notice that:

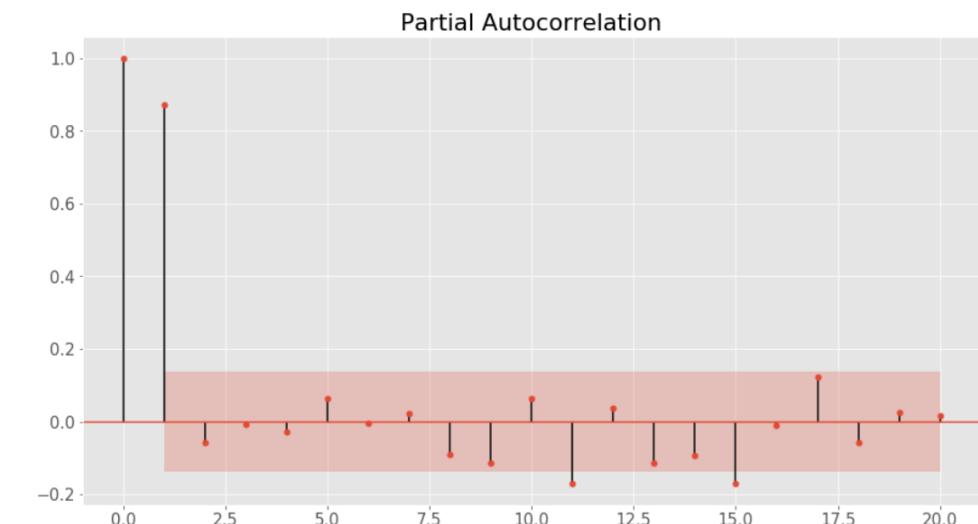
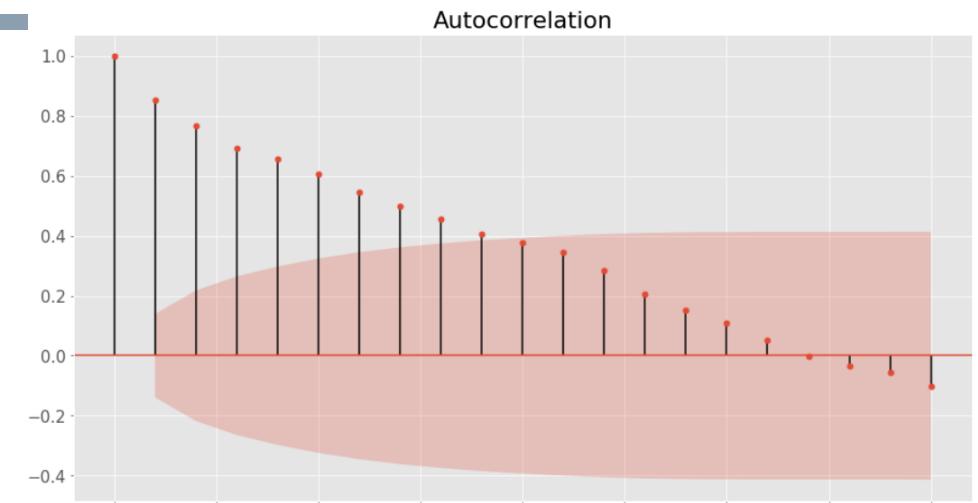
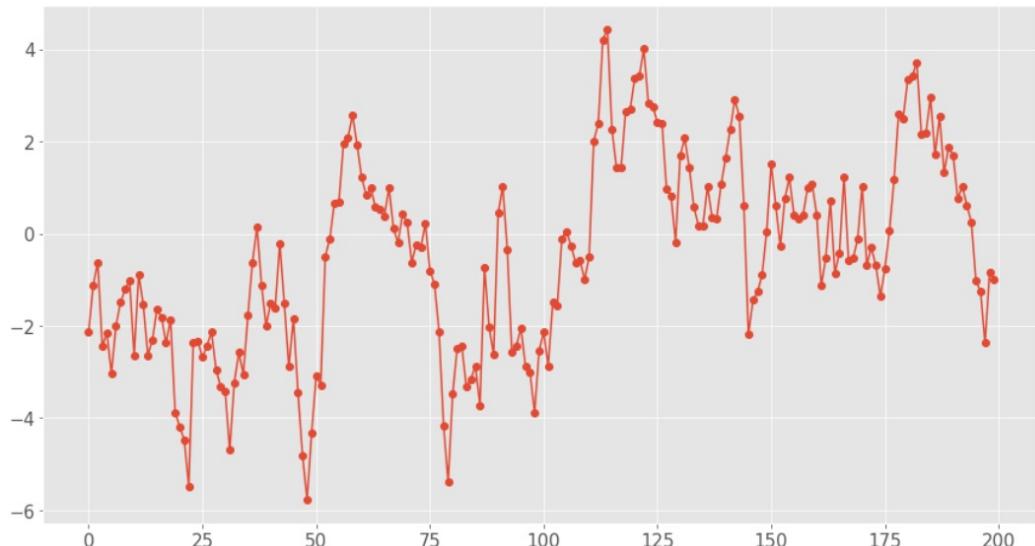
- Only the previous term y_{t-1} and the current noise e_t contribute to the output.
- As $|a_1| \rightarrow 0$, the process looks like white noise.
- When $a_1 < 0$, the process oscillates around zero.
- When $a_1 = 1$, the process is equivalent to a *random walk*.

Example: AR(1)

A numerical example could be given by

$$y_t = 0.9 y_{t-1} + e_t$$

Where, e.g., the Gaussian white noise $e_t = \mathcal{N}(0, 1)$.

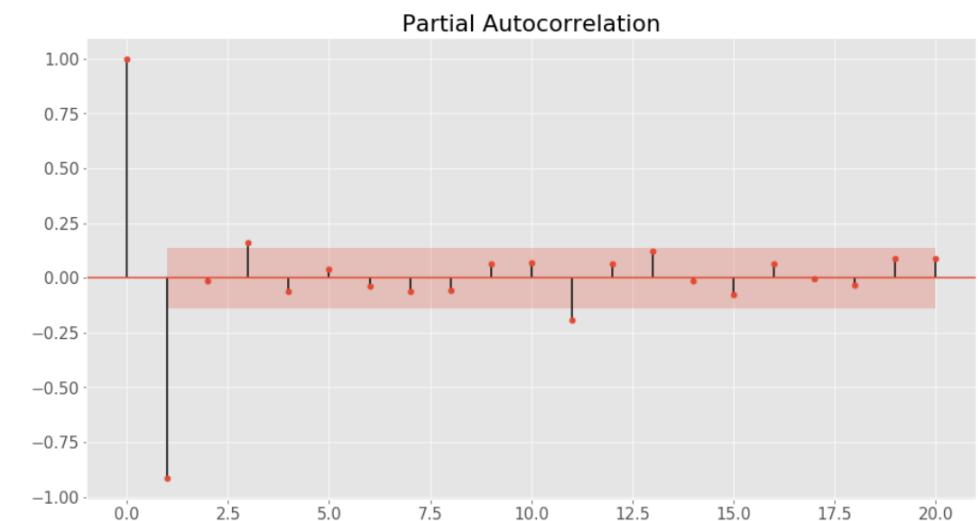
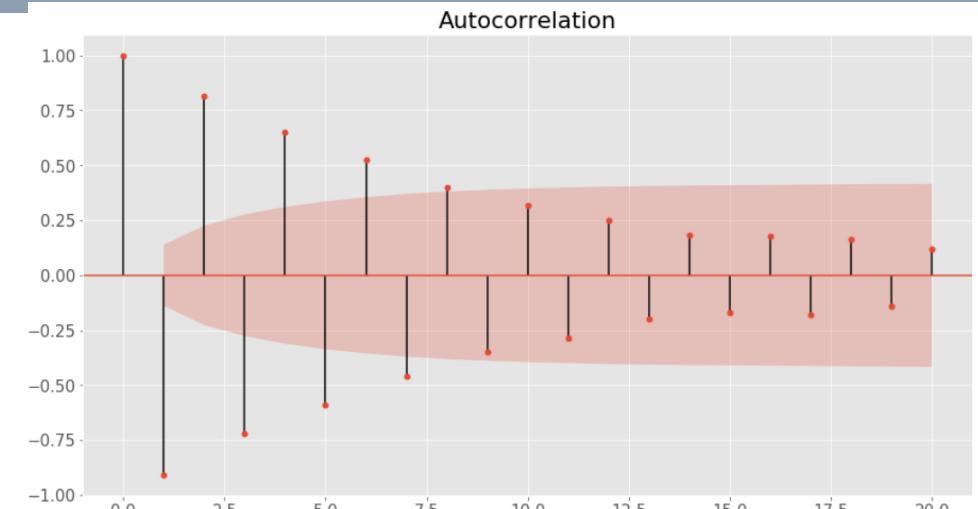
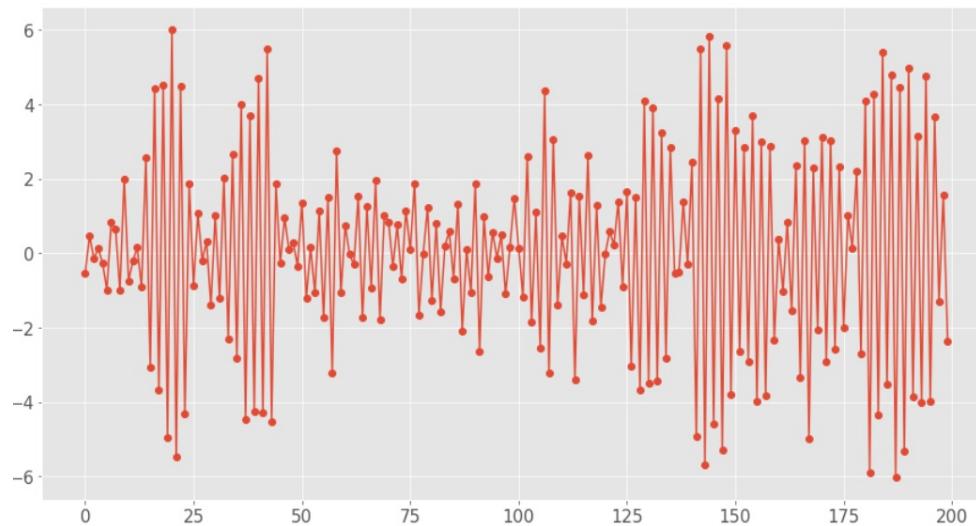


Example: AR(1)

A numerical example could be given by

$$y_t = -0.9 y_{t-1} + e_t$$

Where, e.g., the Gaussian white noise $e_t = \mathcal{N}(0, 1)$.



Chosing an AR(n)

In concrete applications, the value of n is an hyper-parameter to optimize.

It is possible to identify the $AR(n)$ model by looking at the partial autocorrelation function (PACF). In fact:

- The theoretical partial autocorrelation for lags $h > n$ is zero.
→ For concrete experimental data, it might be small but non-zero.
- For $h = n$ the partial autocorrelation ϕ_n is not zero.
→ For all lag values in between, it is not necessarily zero

Moving Average models (MA)

Moving average models (MA) are based on the idea that the value of a time series at time t can be expressed as a linear combination of n past input random shock (or white noise).

$$MA(m): y_t = e_t + b_1 e_{t-1} + \cdots + b_m e_{t-m}$$

Where:

- m is the model's order
- b_1, \dots, b_m are the model's parameters, $b_m \neq 0$

In other words, the hyper-parameter m , again, represents how far back to look for dependencies with previous noise values.

Moving Average models (MA)

Similarly to the autoregressive case, the moving average model $MA(m)$ can be expressed with a more synthetic notation by using the backshift operator:

$$y_t = e_t + b_1 e_{t-1} + \cdots + b_m e_{t-m}$$

$$y_t = (1 + b_1 q^{-1} + \cdots + b_m q^{-m}) e_t$$

$$\mathbf{y}_t = \mathbf{B}(q^{-1}) \mathbf{e}_t$$

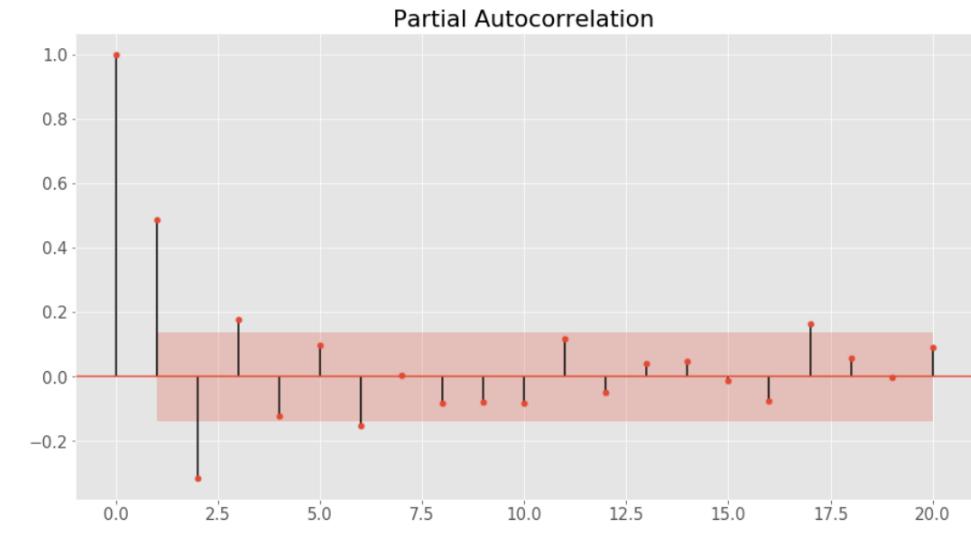
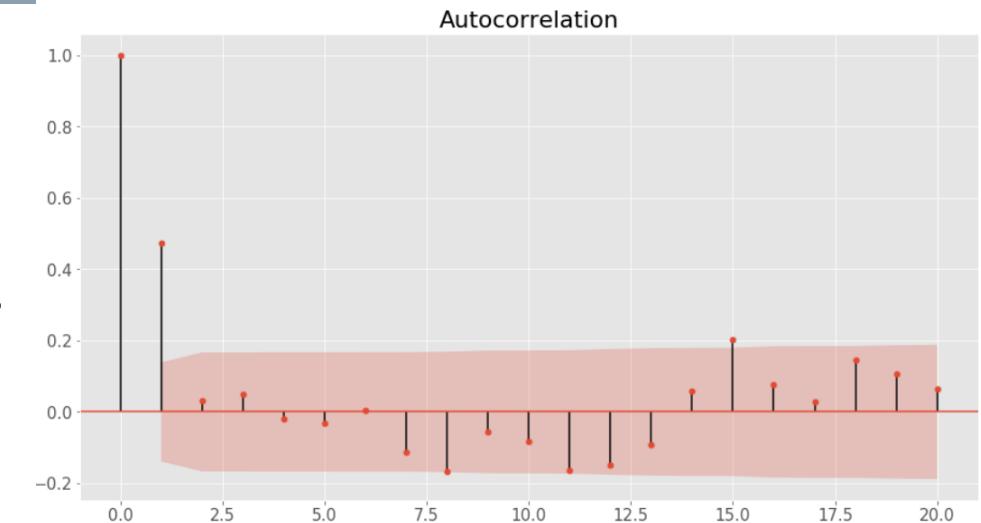
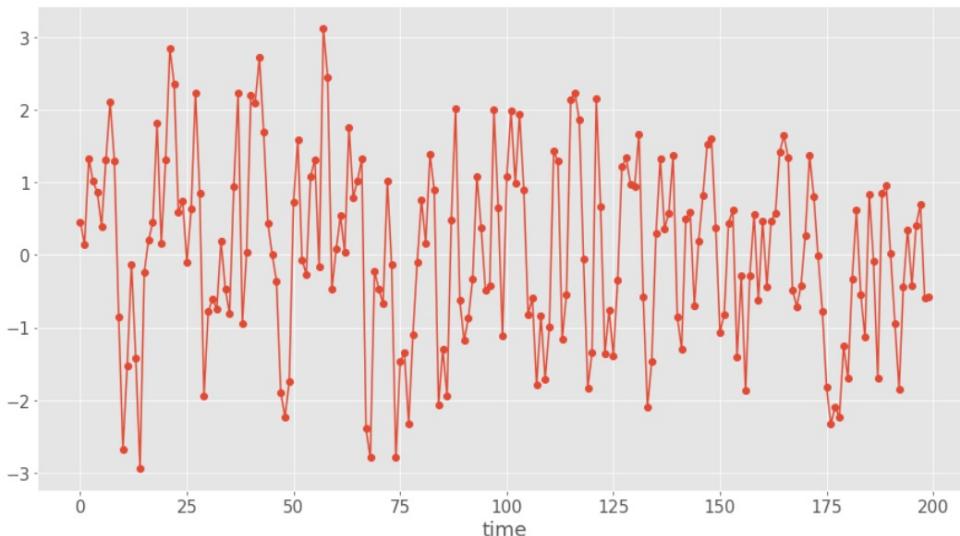
where $\mathbf{B}(q^{-1})$ is called **moving average operator**.

Example: MA(1)

A numerical example could be given by

$$y_t = e_t + 0.8 e_{t-1}$$

Where, e.g., the Gaussian white noise $e_t = \mathcal{N}(0, 1)$.



Chosing an MA(m)

In concrete applications, the value of m is an hyper-parameter to optimize.

It is possible to identify the $MA(m)$ model by looking at the autocorrelation function (ACF).

In fact:

- The theoretical autocorrelation for lags $h > m$ is zero.
→ For concrete experimental data, it might be small but non-zero.
- For $h = m$ the autocorrelation ρ_m is not zero.
→ For all lag values in between, it is not necessarily zero

Critical comparison

- Autoregressive models (AR) ignore correlated noise structures in the time series.
- Differently by AR models, finite moving average models (MA) are always stationary.
- It can be proved that:
 - All finite autoregressive processes $AR(n)$ are infinite moving average processes
 - All finite and invertible moving average $MA(m)$ processes are infinite autoregressive processes
- In practice, parameter estimation for MA models is generally more difficult than for AR models.



Autoregressive models

ARMA and ARIMA models



ARMA models

ARMA model is a combination of **autoregressive (AR)** and **moving average (MA)** models.

$$ARMA(n, m): y_t = a_1 y_{t-1} + a_2 y_{t-2} + \cdots + a_n y_{t-n} + b_1 e_{t-1} + \cdots + b_m e_{t-m} + e_t$$

Which can be re-written using the backshift notation as:

$$ARMA(n, m): A(q^{-1})y_t = B(q^{-1})e_t$$

Where $A(q^{-1})$ is the autoregressive operator and $B(q^{-1})$ is the moving average operator, as defined previously.

Choosing an ARMA(n, m)

We can observe the ACF and PACF to determine the suitable hyper-parameters n and m .

	$AR(n)$	$MA(m)$	$ARMA(n, m)$
ACF	Tails off	Cuts off after lag m	Tails off
PACF	Cuts off after lag n	Tails off	Tails off

The choice of n and m is not unique.

How to deal with Nonstationary time series?

A limitation of the ARMA models is the assumption of our time series to be stationary.

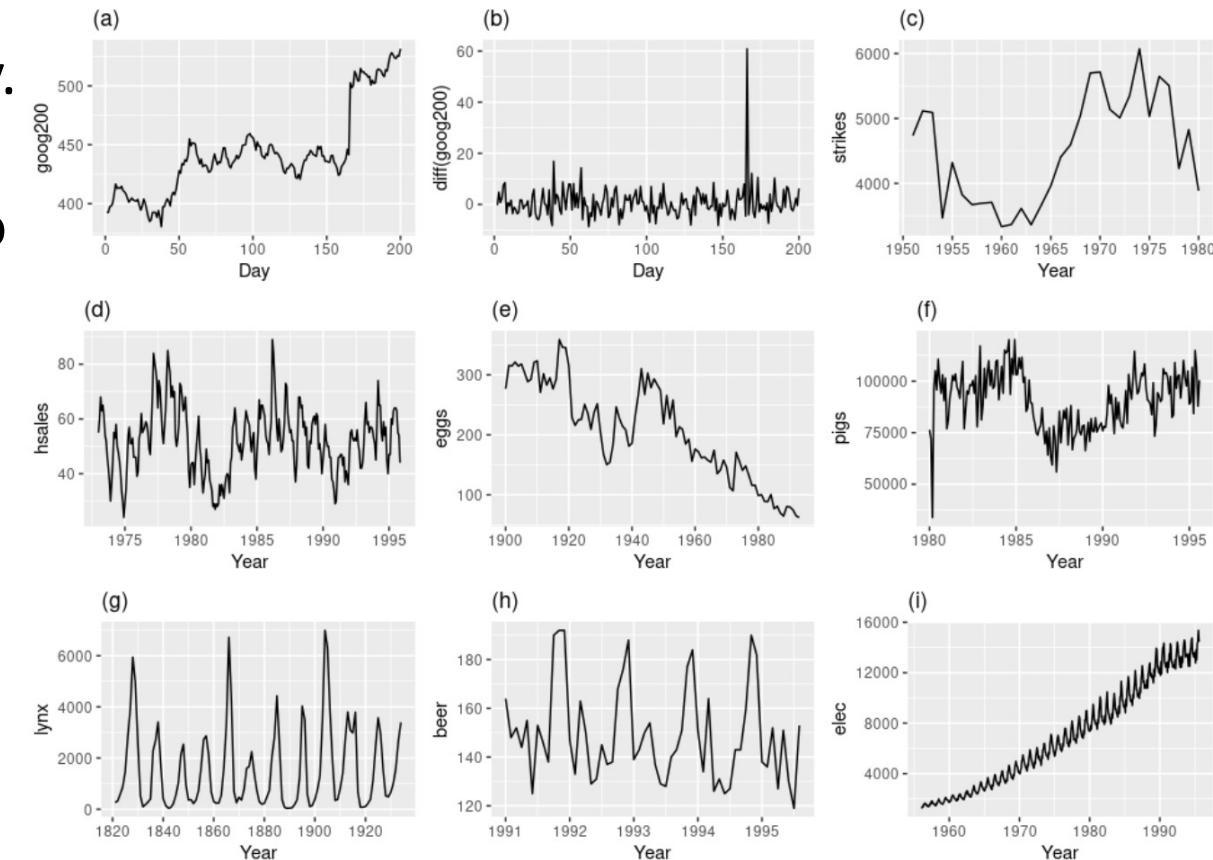
Many times, we can assume the time series to be composed by a **non-stationary trend** and a **zero-mean stationary time series**, i.e.,

$$y_t = \mu_t + \phi_t$$

→ We can „stationarize“ time series.

We can stationarize in two ways:

- Detrending
- Differencing



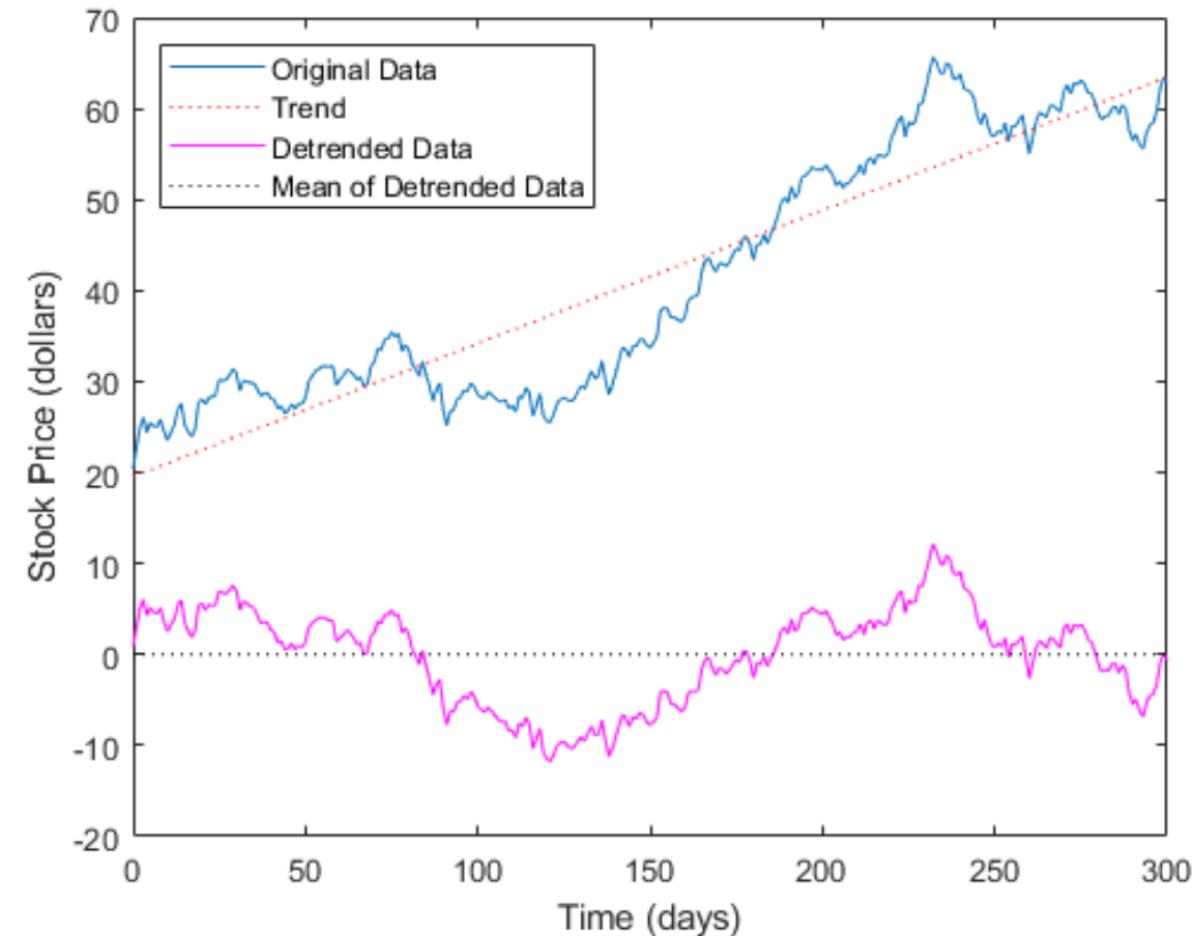
Stationarization: Detrending

By **detrending**, we can subtract an estimate for the time series' trend and deal with the remaining terms (i.e., residuals)

In formulas,

$$\hat{y}_t = y_t - \hat{\mu}_t$$

Detrending needs parameters estimation.



Stationarization: Differencing

The differencing operator is defined by

$$\nabla y_t = y_t - y_{t-1}$$

By using our backshift operator, the operator can be written as

$$\nabla = 1 - q^{-1}$$

Higher-order d differencing operations are given by

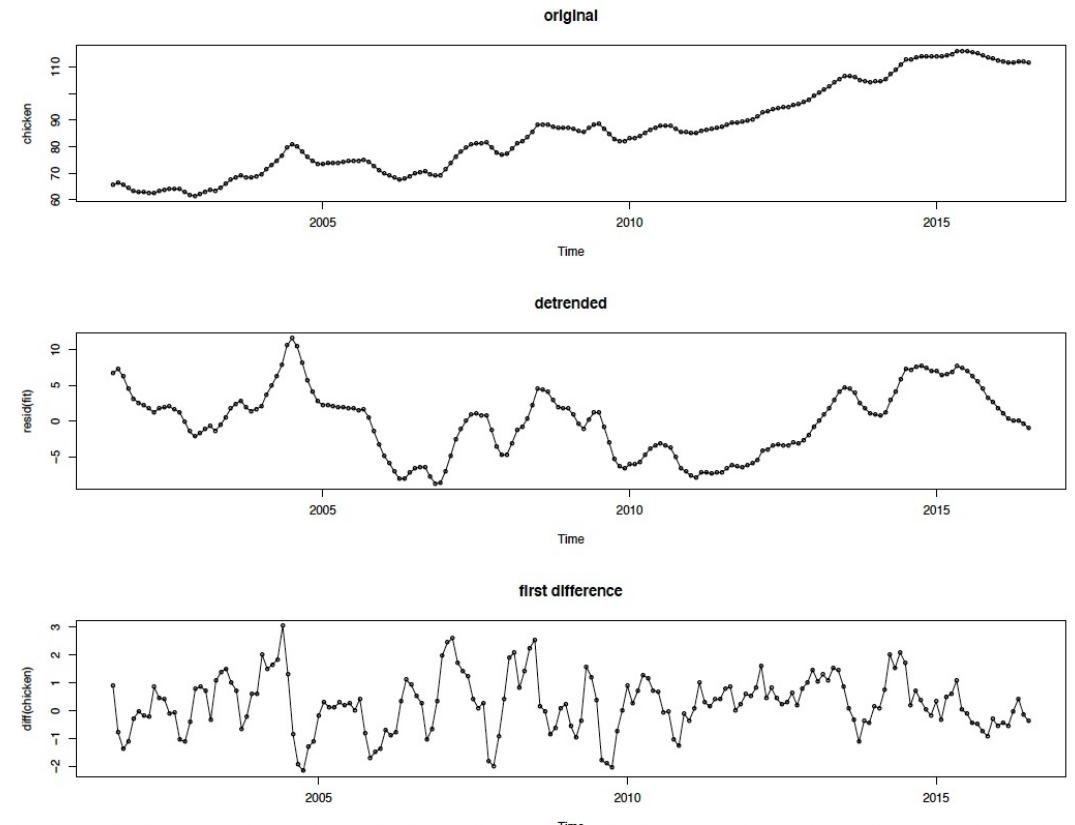
$$\nabla^d = (1 - q^{-1})^d$$

Stationarization: Differencing

By **differencing**, we compute the differences (or higher-order differences) of consecutive observations.

An advantage over detrending is that we do not need to estimate any parameters.

The differencing operation helps to stabilize the mean of a time series, by removing trends and seasonality.



ARIMA models

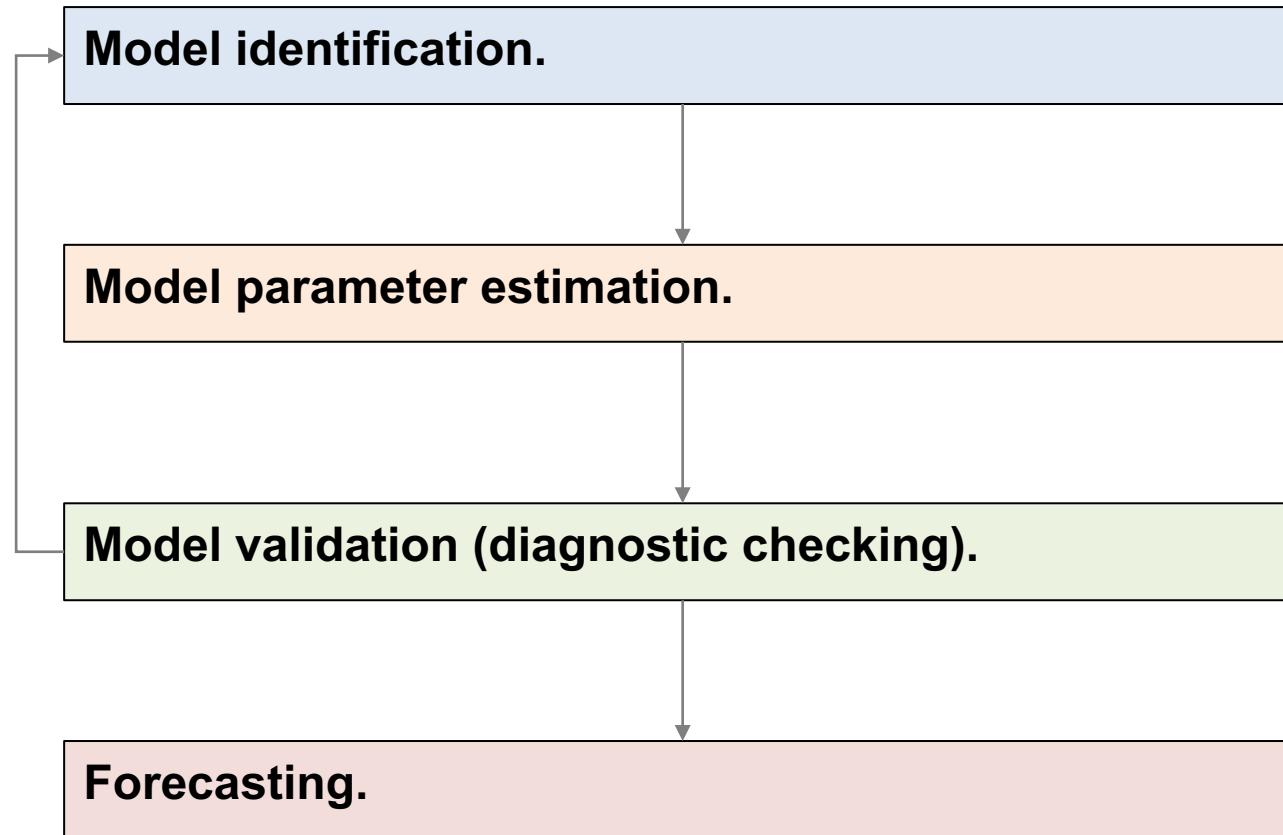
A process y_t is said to be $ARIMA(n, d, m)$ if d -th order differentiation $\nabla^d y_t$ is $ARMA(n, m)$.

Then, the ARIMA model can be written as

$$\textbf{ARIMA}(n, \mathbf{d}, m): A(q^{-1})\nabla^{\mathbf{d}}y_t = B(q^{-1})e_t$$

Notice that, $ARIMA(n, 0, m)$ is equivalent to $ARMA(n, m)$.

General scheme



- Check stationarity and seasonality, perform differentiation if necessary, to chose ARIMA(n, d, m).
- Determine the model's parameters that produce the best fitting, e.g., by Least square (LS) or Maximum likelihood estimation (MLE) methods.
- Perform a diagnostic checking, for example, by residual series analysis.
- We use the selected model for forecasting.



Lecture title

Recap



In this lecture...

- Linear processes
 - Autoregressive processes (AR)
 - Moving average processes (MA)
- Combining AR and MA:
 - ARMA
 - ARIMA

ARIMA: Pros and Cons

- Pros:
 - Effective in short-term series forecasting.
 - E.g., short-run inflation forecasts.
 - It is a parametric model and it works better with relatively small number of observations.
- Cons:
 - Techniques for identifying the correct model are difficult to understand and usually computationally expensive.
 - ARIMA models performance are poor at predicting series with turning points.



Machine Learning for Time Series

(MLTS or MLTS-Deluxe Lectures)

Dr. Dario Zanca

Machine Learning and Data Analytics (MaD) Lab
Friedrich-Alexander-Universität Erlangen-Nürnberg
09.01.2023

- Time series fundamentals and definitions (2 lectures)
- Bayesian Inference (1 lecture)
- Gaussian processes (2 lectures)
- State space models (2 lectures)
- Autoregressive models (1 lecture)
- Data mining on time series (1 lecture) ←
- Deep learning on time series (4 lectures)
- Domain adaptation (1 lecture)

In this lecture...

- 1. Introduction to Data Mining**
- 2. Frequency analysys**
- 3. Dynamic time warping**
- 4. Feature extraction techniques**



Data Mining with Time Series

Data Mining



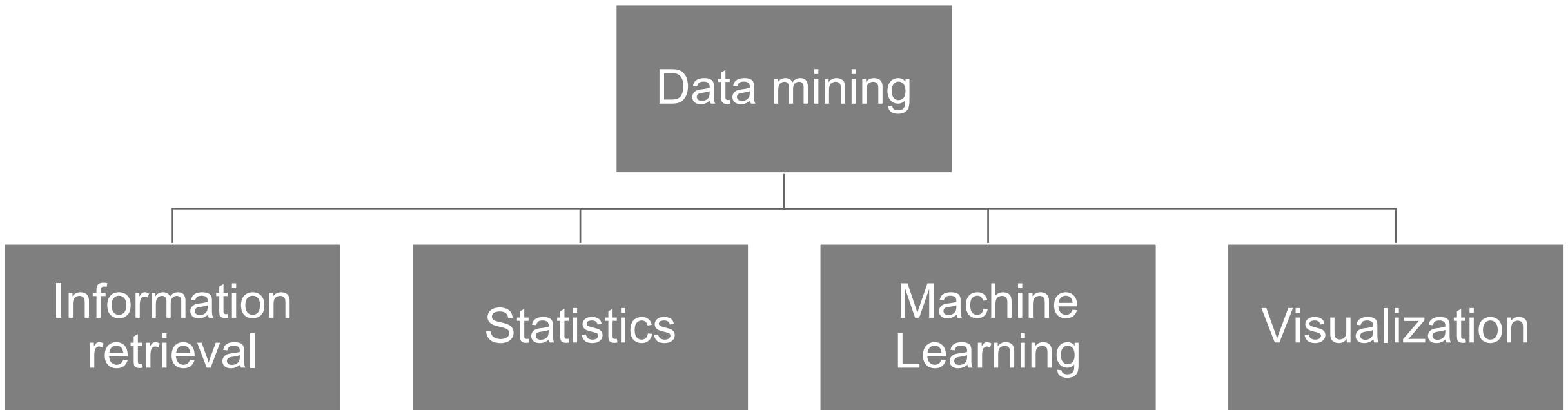
What is Data mining?

Definition 1. Extraction of non-simple, implicit, previously unknown, possibly useful data from database.

Definition 2. Automatic or semi-automatic search and analysis of a large amount of data with the goal of discovering significant patterns.

Data mining is useful when the information is hidden due to large amount of data, its complexity, heterogeneity, the speed at which it is collected and need for non-traditional queries.

What is Data mining?



Data mining

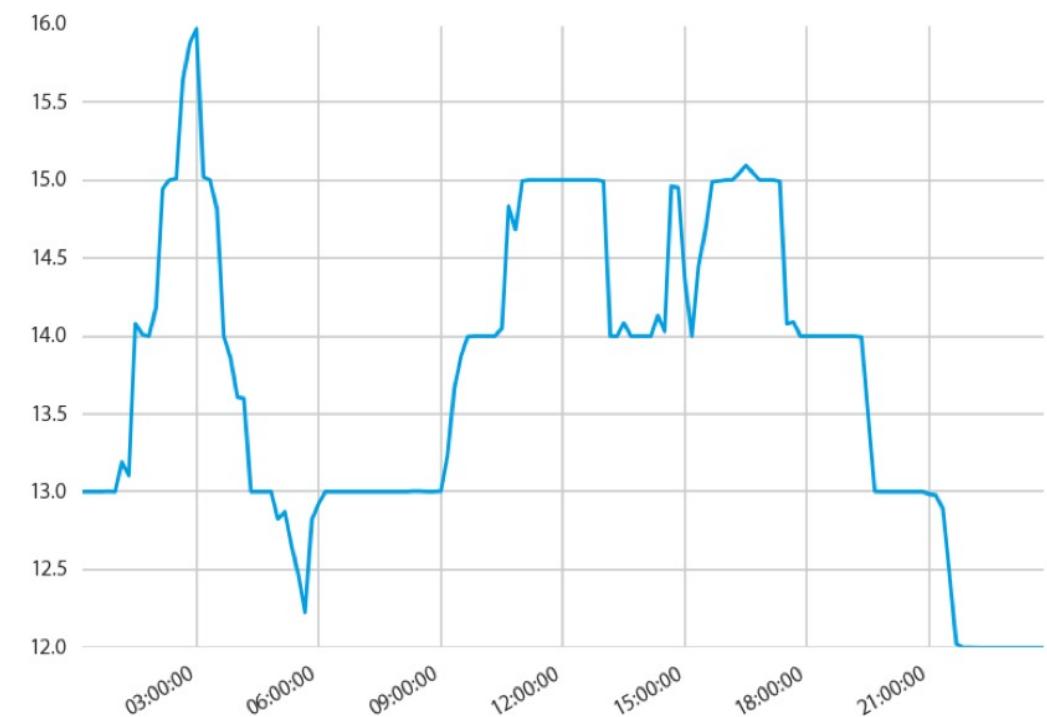
The most basic approach for data mining can include:

- **Compute data characteristics**
- **Data reduction**
- **Data transformation**

Compute data characteristics

Suppose we have a time series of ambient temperature recorded during one day.

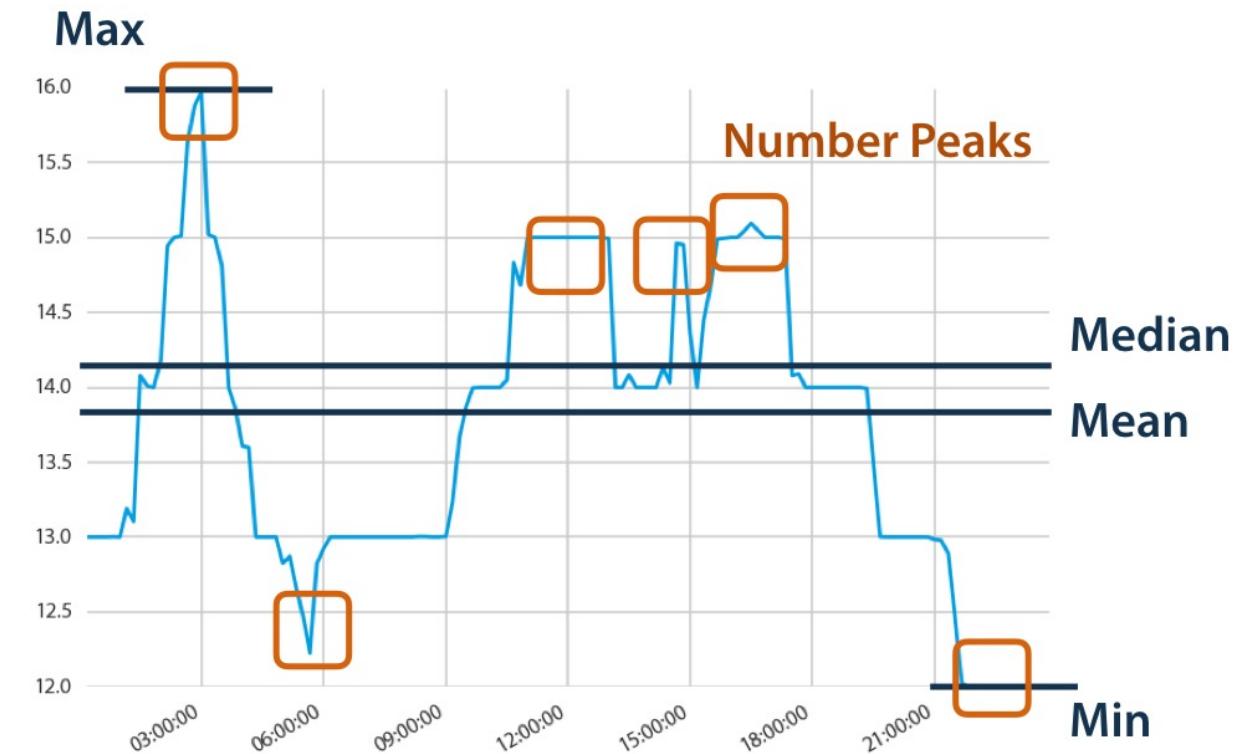
A characterization of this time series can be given by a set of **basic statistics**.



Compute data characteristics

Suppose we have a time series of ambient temperature recorded during one day.

A characterization of this time series can be given by a set of **basic statistics**.



Data reduction

When dealing with big dataset, the application of **data reduction techniques** is required in order to allow reasoning on smaller dimensional spaces.

- Efficiency
- Interpretability
- Simplicity

There are three main ways to reduce data size:

- **Sampling.** Reducing the number of observations.
- **Selection and projection.** Reducing the number of features.
- **Discretization and aggregation.** Reduction of the number of possible values.

Data transformation

In most applications, it is necessary to **apply transformations to our data** to make it usable for further analysis.

- Most machine learning algorithms perform better if data has a consistent scale distribution.
- E.g., data is heterogenous because of different data sources (units of measure, sampling rates, data types).

Different data transformations:

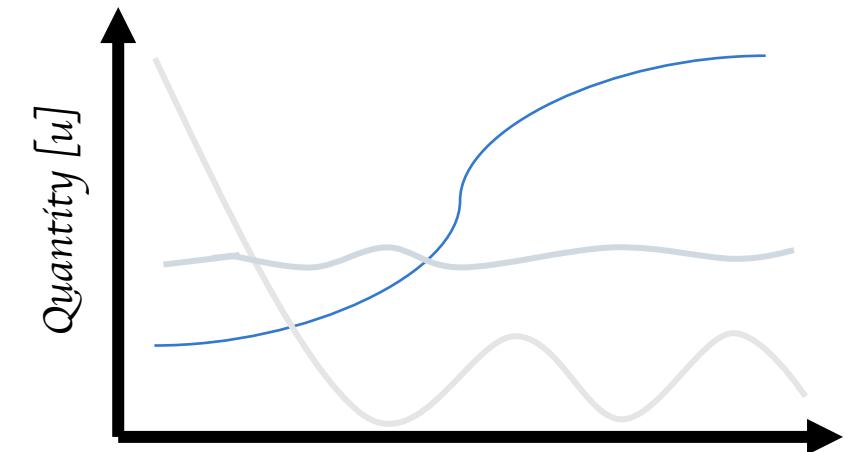
- Normalization
- Standardization

Data transformation: Normalization

Feature normalization is used to normalize the range of values of features.

Methods of data normalization:

- Mean normalization
- Min-Max normalization



Data transformation: Mean Normalization

Let $S = (s_1, \dots, s_T)$ be a multivariate time series, $s_i \in \mathbb{R}^d$ is a d -dimensional observation at time t_i .

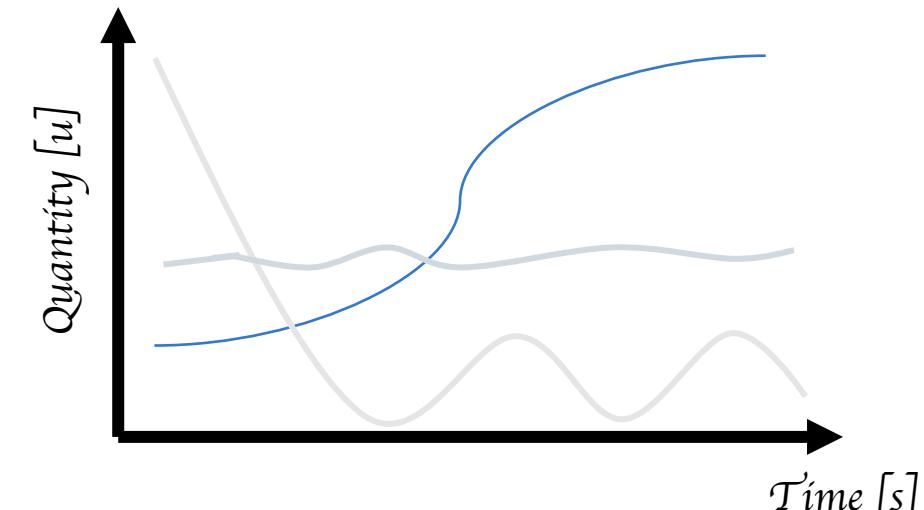
We denote with $\mathbf{S}_j = (s_{1j}, \dots, s_{Tj})$ the j -th feature of the time series S , where $s_{ij} \in \mathbb{R}$ is an observation of the j -th feature at time t_i .

Mean normalization is defined by:

$$s'_{ij} = \frac{s_{ij} - \mu_j}{s_{\max,j} - s_{\min,j}}$$

where $s_{\max,j}$ and $s_{\min,j}$ are the max and min values of \mathbf{S}_j ,

and $\mu_j = \frac{1}{T} \sum_{i=1}^T s_{ij}$.

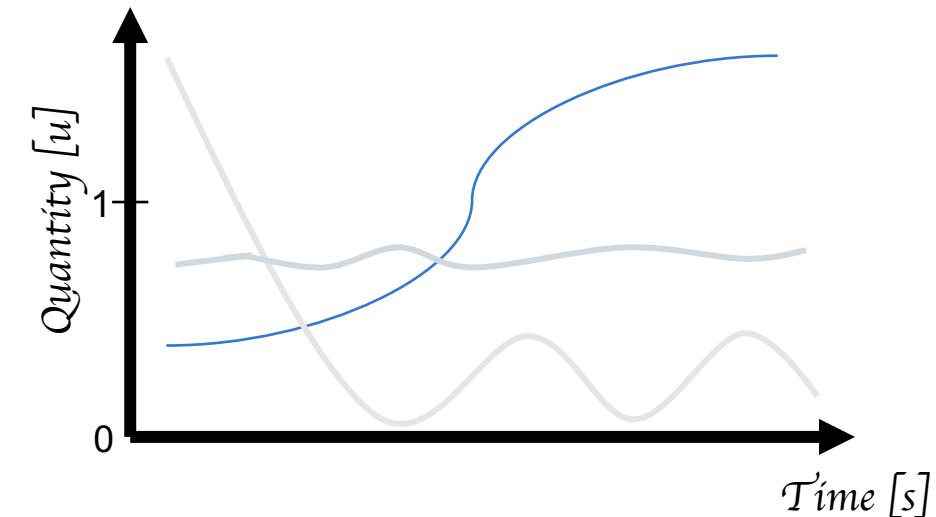


Data transformation: Min-Max Normalization

Min-Max normalization is defined by:

$$s'_{ij} = \frac{s_{ij} - s_{\min,j}}{s_{\max,j} - s_{\min,j}}$$

where $s_{\max,j}$ and $s_{\min,j}$ are the max and min values of S_j .



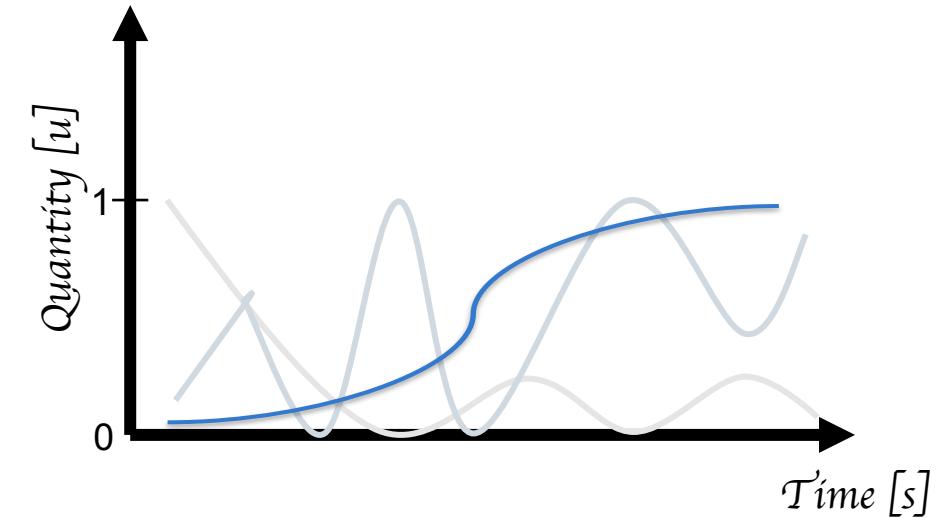
Data transformation: Min-Max Normalization

Min-Max normalization is defined by:

$$s'_{ij} = \frac{s_{ij} - s_{\min,j}}{s_{\max,j} - s_{\min,j}}$$

where $s_{\max,j}$ and $s_{\min,j}$ are the max and min values of S_j .

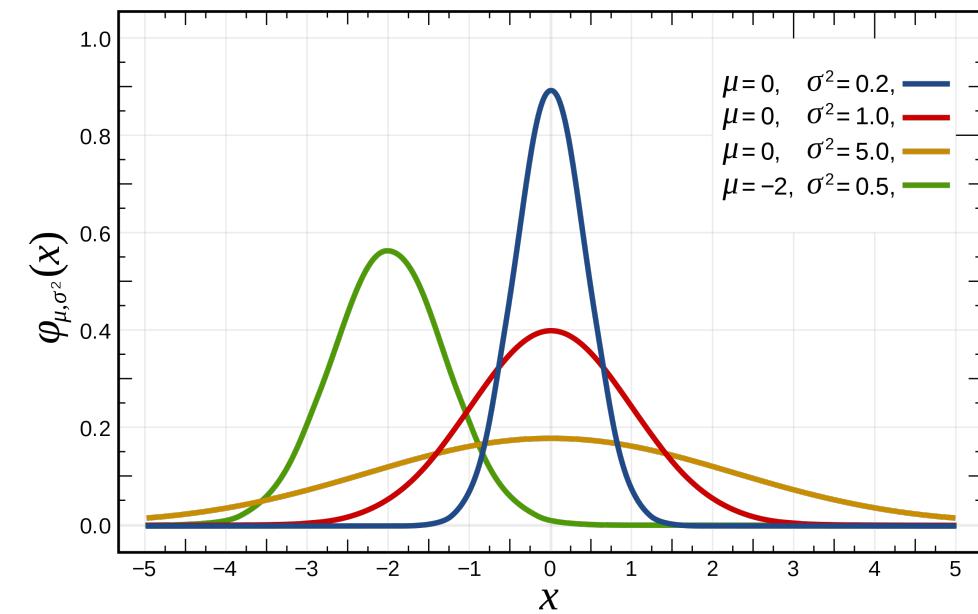
This normalization establish two boundaries for the data between 0 and 1.



Data transformation: Standardization

Standardization is the process of converting a random variable with mean μ and standard deviation σ to a “**standard distribution**” (i.e., zero mean and unit standard deviation).

The **standard score** is the number of standard deviations by which the value of a raw score (i.e., an observed value or data point) is above or below the mean value of what is being observed or measured.



https://en.wikipedia.org/wiki/Normal_distribution

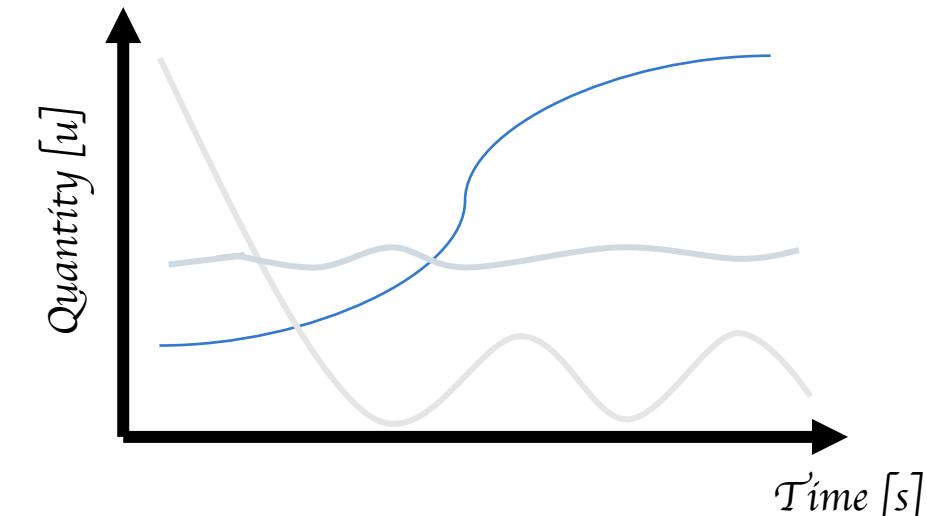
Data transformation: Z-score standardization

Z-score standardization is defined by:

$$s'_{ij} = \frac{s_{ij} - \mu_j}{\sigma_j}$$

where $s_{\max,j}$ and $s_{\min,j}$ are the max and min values of S_j , and μ_j and σ_j are the feature mean and standard deviation.

- The transformed data will have zero mean and unit standard deviation.
- It is empirically shown that, if the original distribution is Gaussian, z-score based transformation generates values that are in the range $(-3, 3)$.





Data Mining with Time Series

Frequency analysis



Spectral analysis

Spectral analysis is a technique that allows us to discover underlying periodicities.

- Many time series show periodic behavior.
- This periodic behavior can be very complex.
- Additional tool to analyse time series (complementary to the time-domain analysis)

To perform spectral analysis, we first must transform data from time domain to frequency domain.

→ We use Fourier transform to convert the time series from the time-domain to the frequency domain

Fourier representation

Given a time series $S = (x_1, \dots, x_N)$, the goal of spectral analysis is that of determining how to construct it using sines and cosines, i.e.,

$$S = \sum_k a_k \sin\left(2\pi \frac{k}{N} t\right) + b_k \cos\left(2\pi \frac{k}{N} t\right)$$

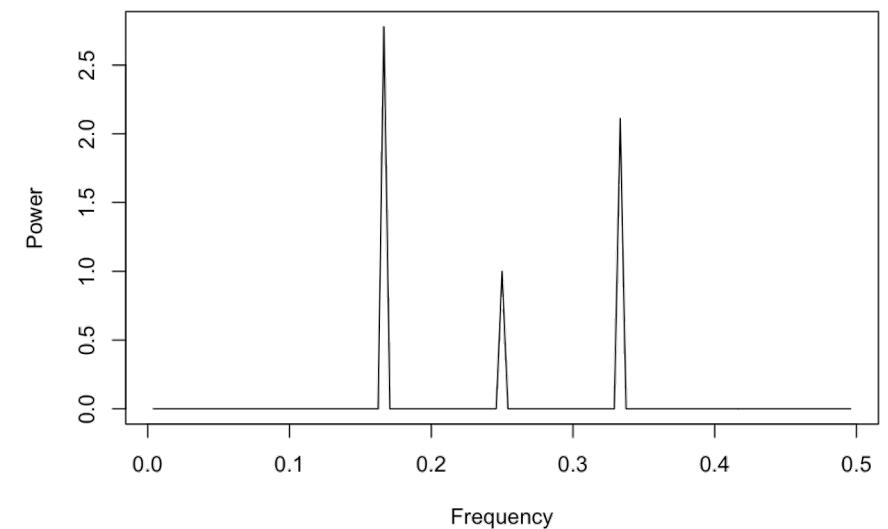
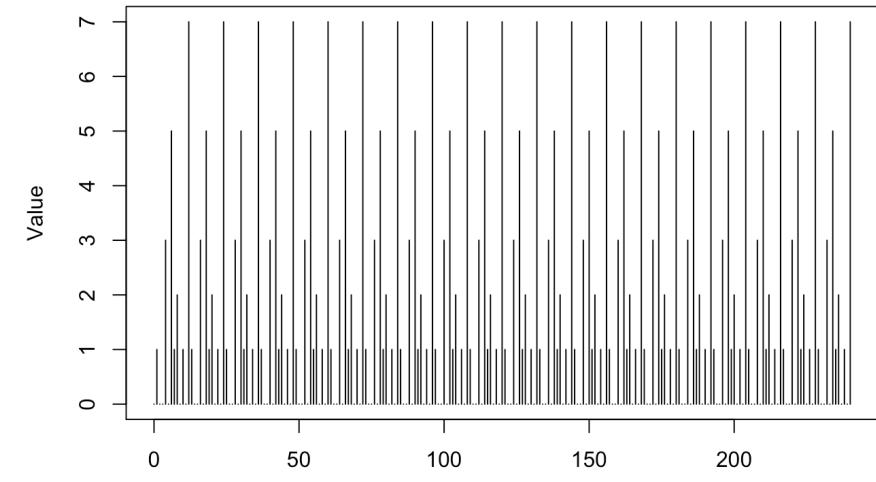
The above expression is called **Fourier representation** for a time series.

- It allows us to re-express time series in a standard way
- Different time series are characterized by different coefficients
 - a_k 's and b_k 's can be determined in a closed form
 - We can compare time series by comparing their coefficients

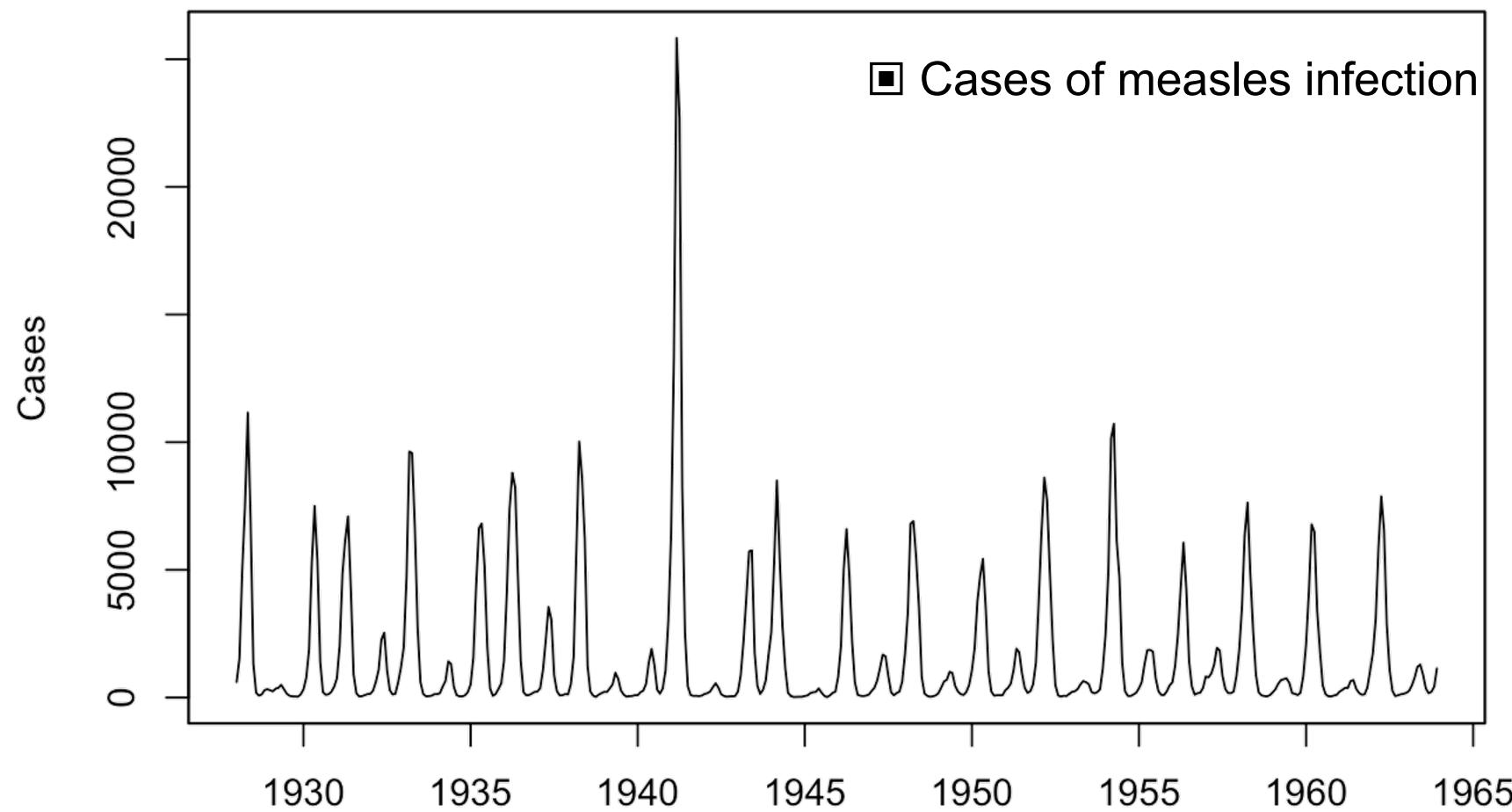
Spectral density

In brief, the covariance of the time series can be represented by a function known as the **spectral density**.

The spectral density can be estimated using an object known as a **periodogram**, which is the squared correlation between our time series and sine/cosine waves at the different frequencies spanned by the series.



Example: spectral analysis

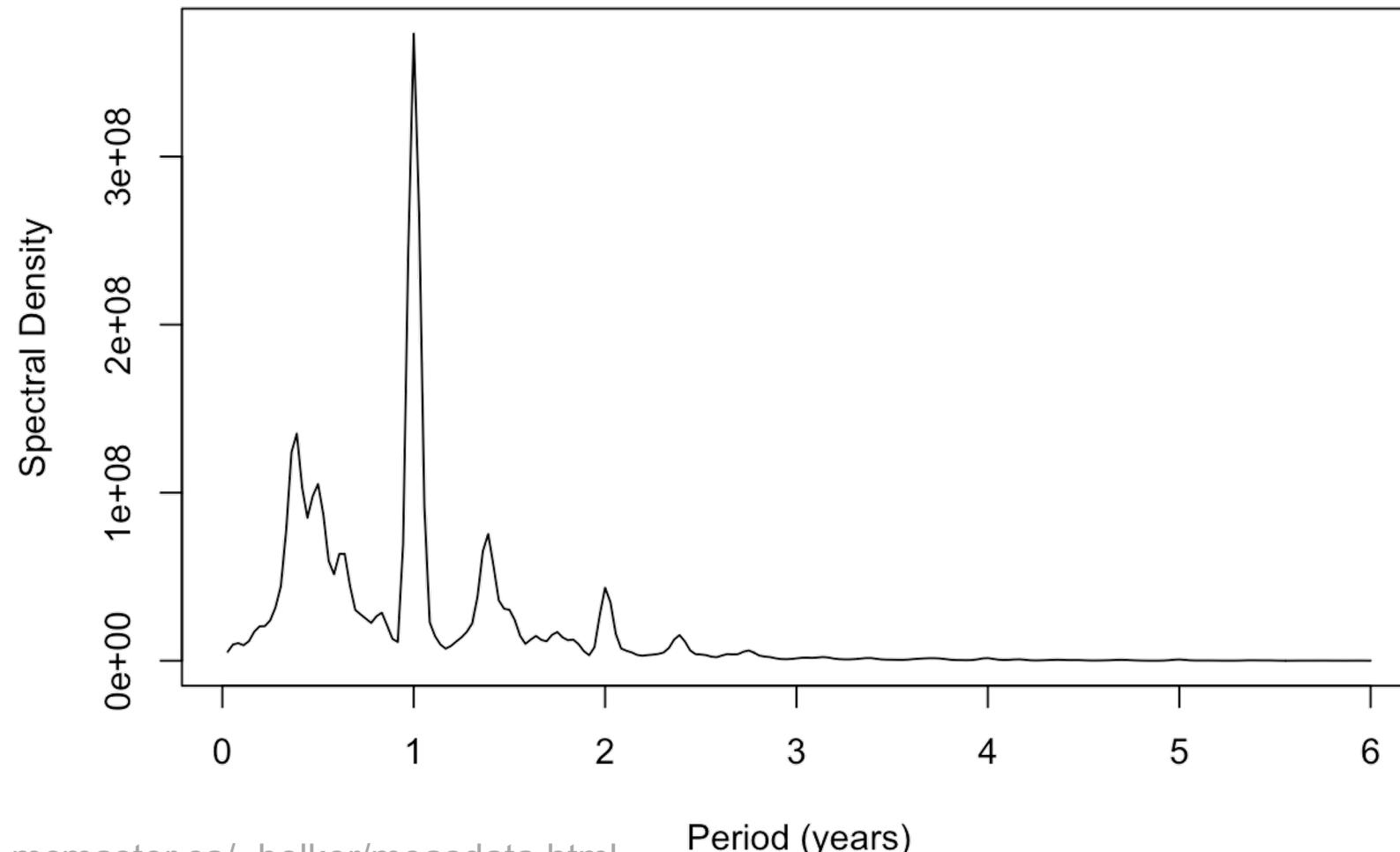


Data from: <https://ms.mcmaster.ca/~bolker/measdata.html>

Date

Images from: <http://web.stanford.edu/class/earthsys214/notes/series.html#spectral-analysis>

Example: spectral analysis



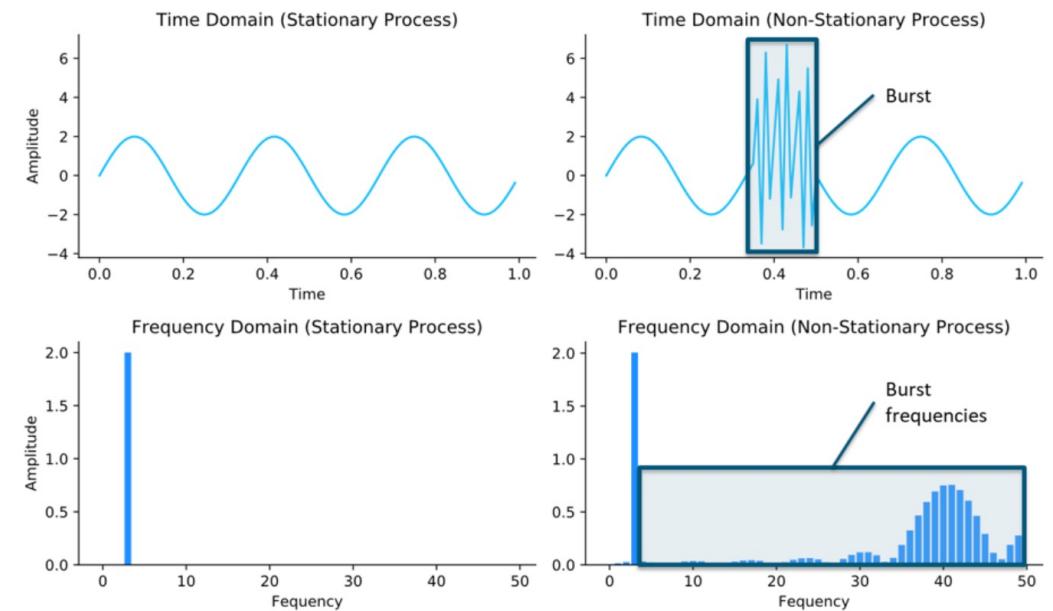
Data from: <https://ms.mcmaster.ca/~bolker/measdata.html>

Images from: <http://web.stanford.edu/class/earthsys214/notes/series.html#spectral-analysis>

Limitations of the Fourier transform

Fourier transform works well for sines/cosines waves which are generated by stationary signals.

For a non-stationary signal (e.g., containing an burst /anomaly) by performing a Fourier transform we obtain frequencies that construct the signal but we cannot identify which of them represents the burst/anomaly.

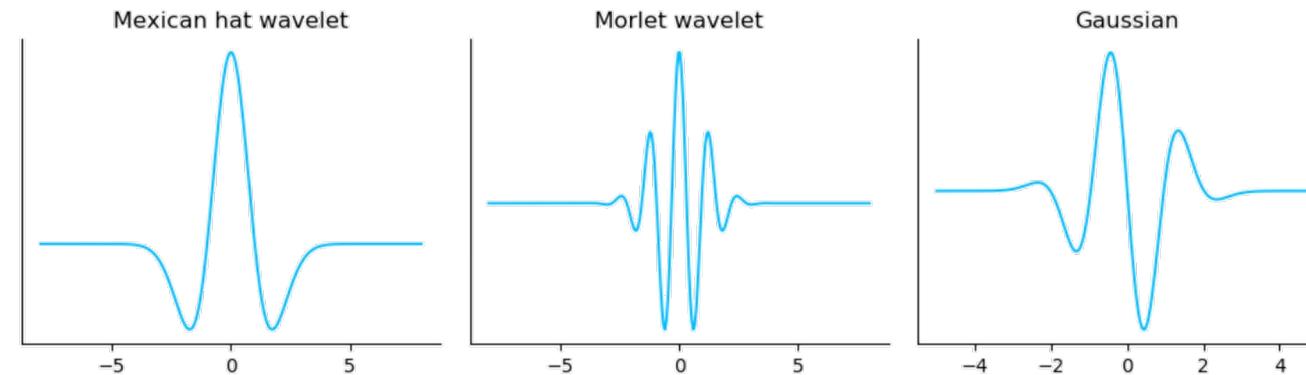


<https://towardsdatascience.com/multiple-time-series-classification-by-using-continuous-wavelet-transformation-d29df97c0442>

→ With non-stationary signlas we are confined to either time or frequency domain.

Continuous Wavelet Transform

Continuous Wavelet Transform (CWT) is based on the concept of **wavelets** (or mini wavelets).



- In contrast to sines/cosines used in Fourier transform, wavelets are limited (in time)
- Have zero mean.

Both these conditions allow a localization in time and frequency at the same time.

Continuous Wavelet Transform

Continuous Wavelet Transform (CWT) is defined as follows:

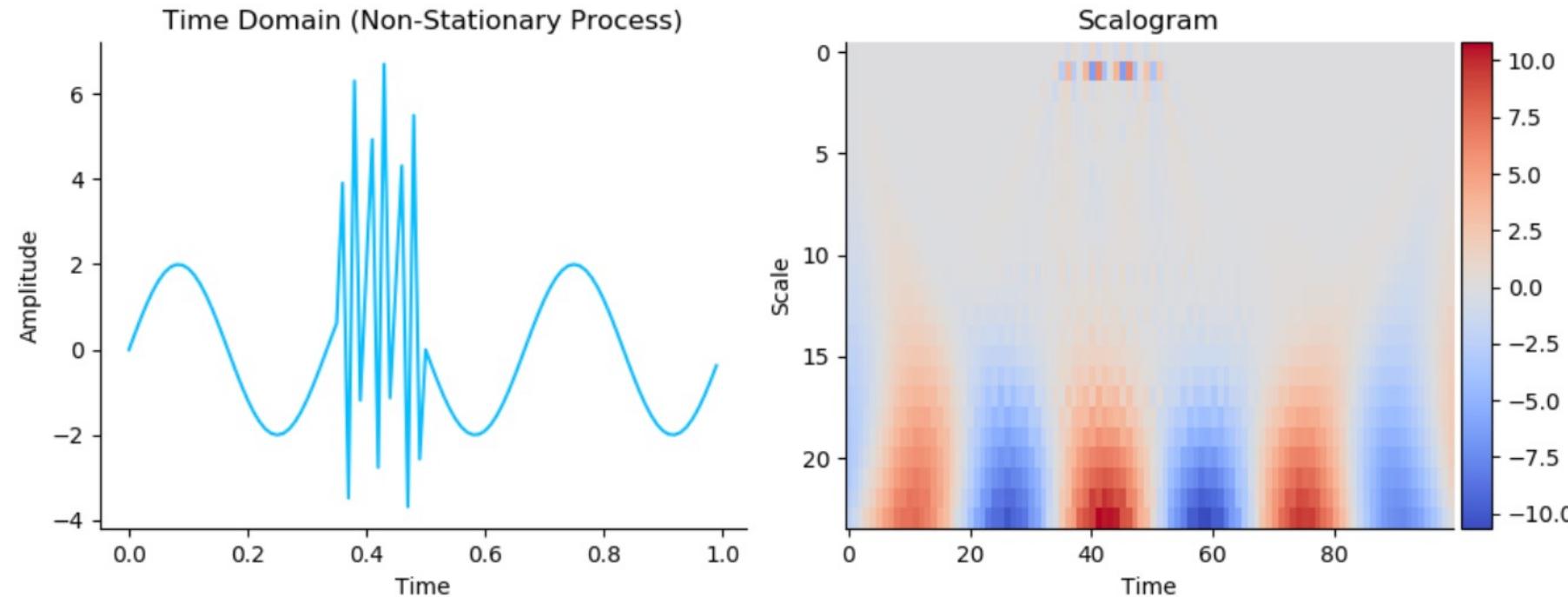
$$cwt(\tau, s) = \frac{1}{\sqrt{|s|}} \int_{-\infty}^{+\infty} x(t)\psi\left(\frac{t - \tau}{s}\right) dt$$

where τ is the translation, s is the scale, ψ is the mother wavelet.

- The CWT is based on the concepts of scaling and shifting.
- It transforms the original 1-D time series to a N-D "image".

Example: Continuous Wavelet Transform

If we apply CWT to the previous non-stationary signal, we obtain:



Continuous Wavelet Transform

Continuous Wavelet Transform (CWT) are:

- Efficient in determining the damping ratio of oscillating signals (e.g. identification of damping in dynamic systems)
- Robust to the noise in the signal
- 2D scalogram can be used to improve the distinction between varying types of a signal.

E.g.,

- differentiate between different production processes in a machine
- identifying components or tools faults



Data Mining with Time Series

Dynamic Time Warping

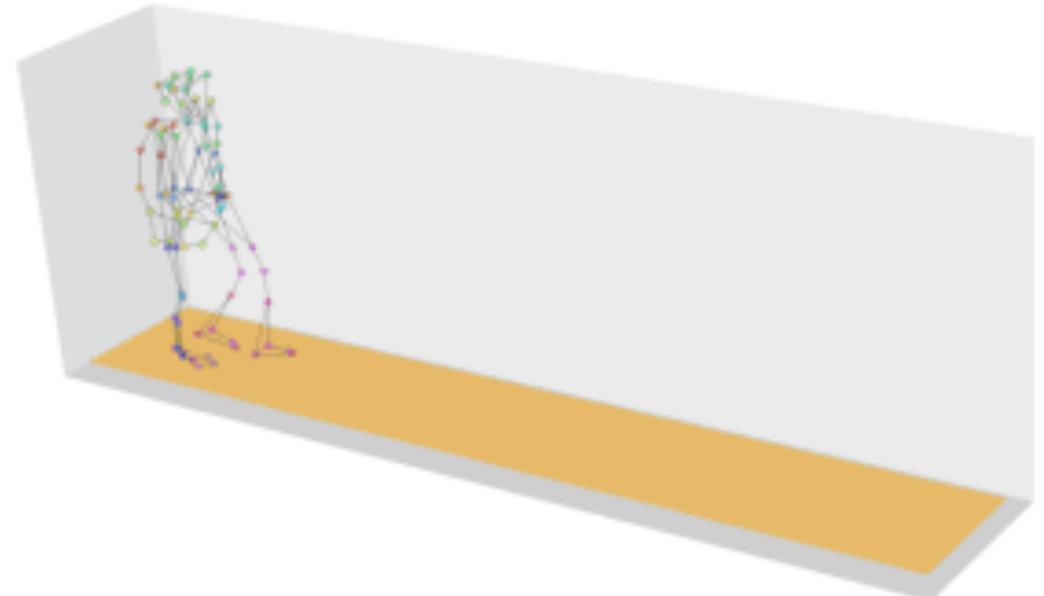


Dynamic time warping

In many application, there is the need to analyse multiple time series at the same time to find similarities between time series.

- E.g., speech recognition, signature recognition, similarity in walking, ...

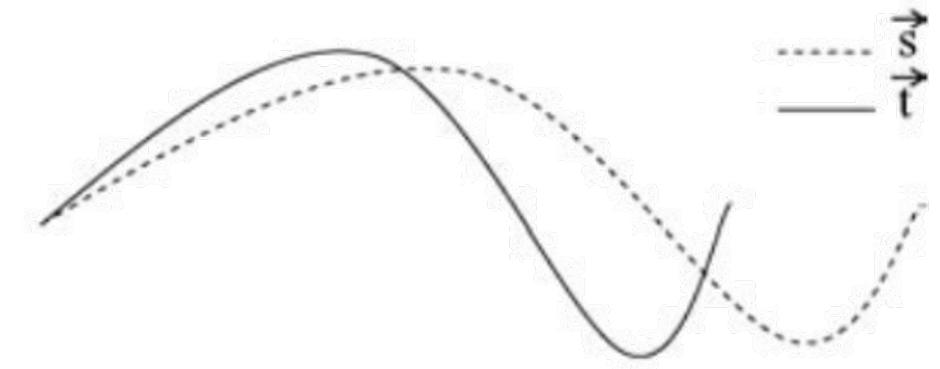
Euclidean distance does not work for time series that are not perfectly synchronized.



Dynamic time warping

Dynamic time warping (DTW) is an algorithm to measure similarity between two time-series that may **vary in speed and time**.

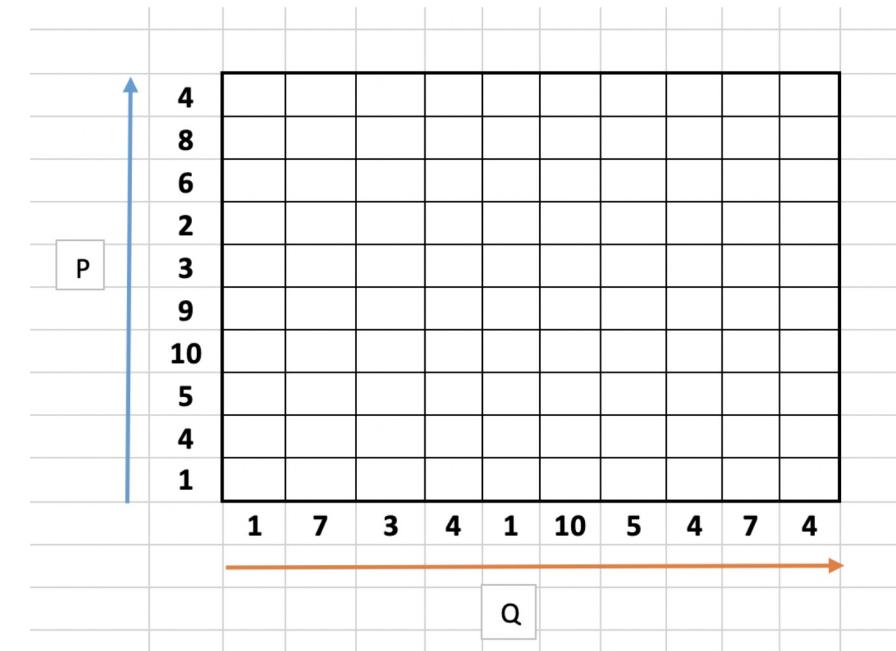
DTW determines the optimal global alignment between two time series.



Dynamic time warping: Algorithm

Given two time series x_t and y_t , we can compute the DTW distance as follows:

1. Initialize the distance matrix M



Dynamic time warping: Algorithm

Given two time series x_t and y_t , we can compute the DTW distance as follows:

1. Initialize the distance matrix M
2. Fill M from the bottom left corner,
according to the formula:

$$M(i, j) = \text{dist}(x_i, y_j) + \min(M(i - 1, j - 1), M(i, j - 1), M(i - 1, j))$$

Please notice:

- $M(i, j)$ denotes the cell of the i -th row and j -th column in the M matrix, starting from the bottom left.
- Near to the axis, $M(i - 1, j - 1)$, $M(i, j - 1)$ and $M(i - 1, j)$ fall “outside” the matrix M and are considered as “0” in the equation.

Dynamic time warping: Algorithm

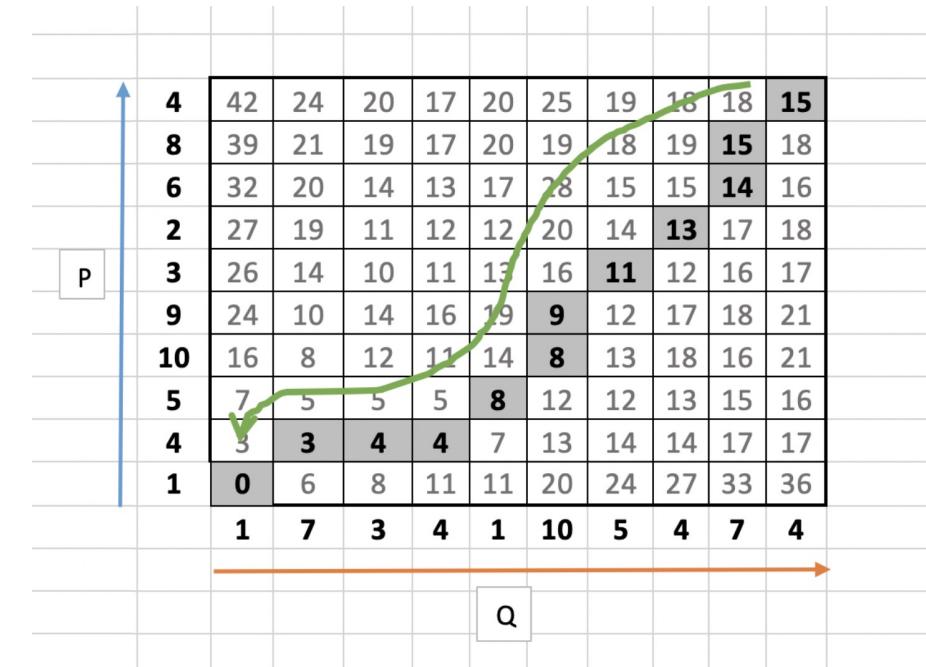
Given two time series x_t and y_t , we can compute the DTW distance as follows:

1. Initialize the distance matrix M
2. Fill M from the bottom left corner,

according to the formula:

$$M(i, j) = \text{dist}(x_i, y_j) + \min(M(i - 1, j - 1), M(i, j - 1), M(i - 1, j))$$

3. Identify the warping path d , starting from the top right corner



Dynamic time warping: Algorithm

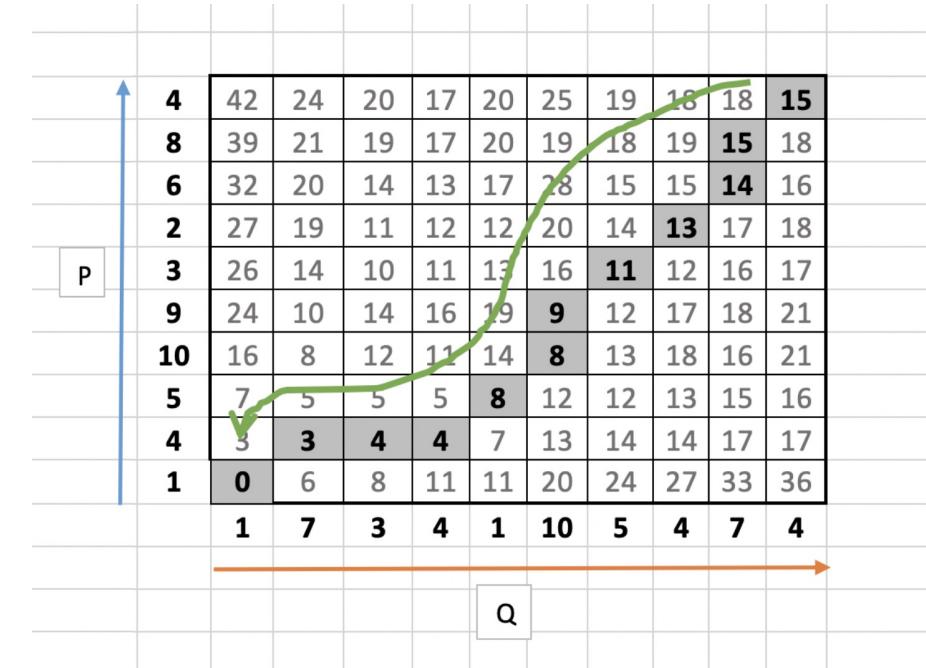
Given two time series x_t and y_t , we can compute the DTW distance as follows:

1. Initialize the distance matrix M
2. Fill M from the bottom left corner, according to the formula:

$$M(i, j) = \text{dist}(x_i, y_j) + \min(M(i - 1, j - 1), M(i, j - 1), M(i - 1, j))$$

3. Identify the **warping path** d , starting from the top right corner
4. The overall path cost can be calculated as

$$D = \sum_{i=1}^k \text{dist}(i_k, j_k)$$



Dynamic time warping: Pros and Cons

Pros:

- Exploit a non-linear distortion (in time) to find non-trivial similarity

Cons:

- **High computational cost.** Alternatives for computing the alignment path more efficiently have been presented.
- It needs the preparation of reliable reference templates for the set of words to be recognized.



Data Mining with Time Series

Other feature extraction methods for time series



Similarity join

One method to find anomalies and trends in time series is to perform a **similarity join**.

- Compare pair of snippets in a time series
- Retrieve all data pairs whose distances are smaller than a predefined threshold ϵ .
- **Very easy to implement**
- **Computationally expensive for moderately large collection of data**

Matrix profiles

Matrix profile (MP) is a data structure for time series analysis.

Advantages:

- MP is domain agnostic
- Efficient
- Provides exact solution
- Only requires a single parameter

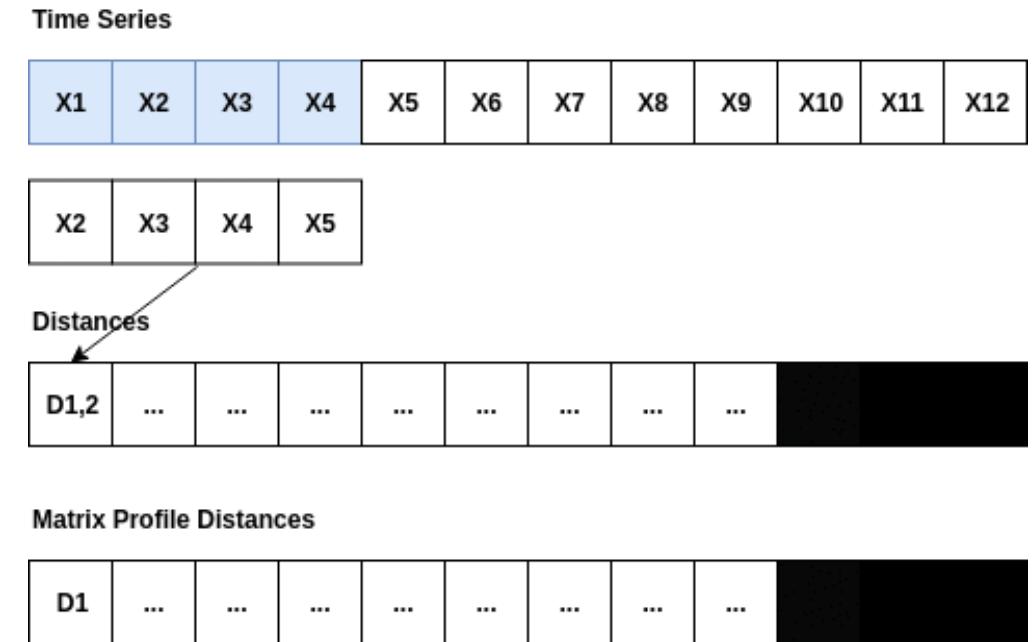
Matrix profiles

The Matrix Profile has two primary components:

- **distance profile**: a vector of minimum Z-Normalized Euclidean distances
- **profile index**: it contains the index of its first nearest-neighbor

The algorithms that compute the Matrix Profile use a **sliding window** approach.

Let m be the window size, and n be the time series length.



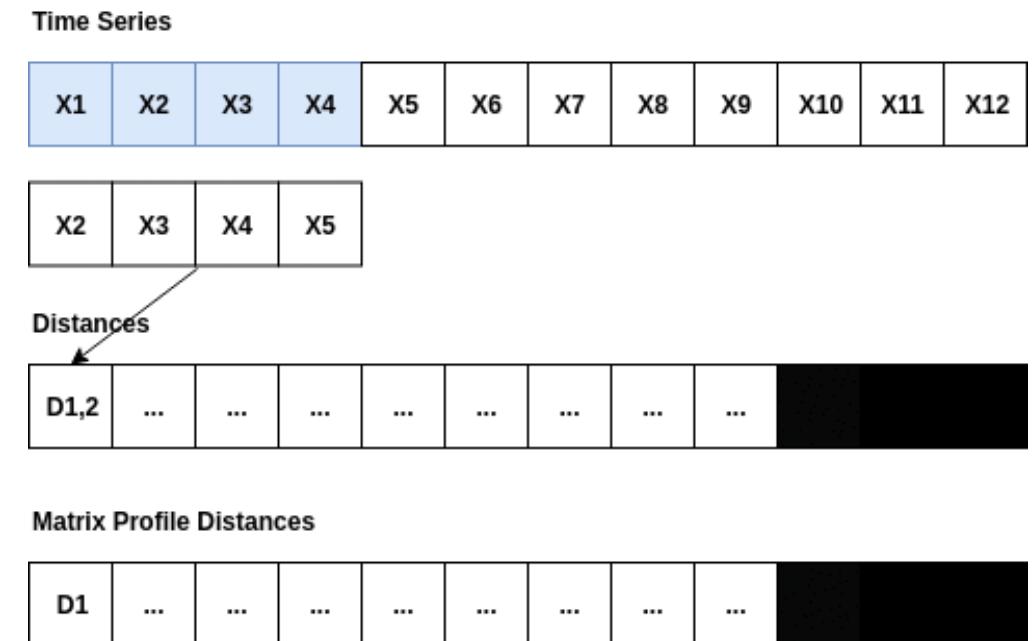
<https://towardsdatascience.com/introduction-to-matrix-profiles-5568f3375d90>

Original paper: All Pairs Similarity Joins for Time Series: A Unifying View that Includes Motifs, Discords and Shapelets. Chin-Chia Michael Yeh, Yan Zhu, Liudmila Ulanova, Nurjahan Begum, Yifei Ding, Hoang Anh Dau, Diego Furtado Silva, Abdullah Mueen, Eamonn Keogh (2016). IEEE ICDM 2016.

Matrix profiles: the algorithm

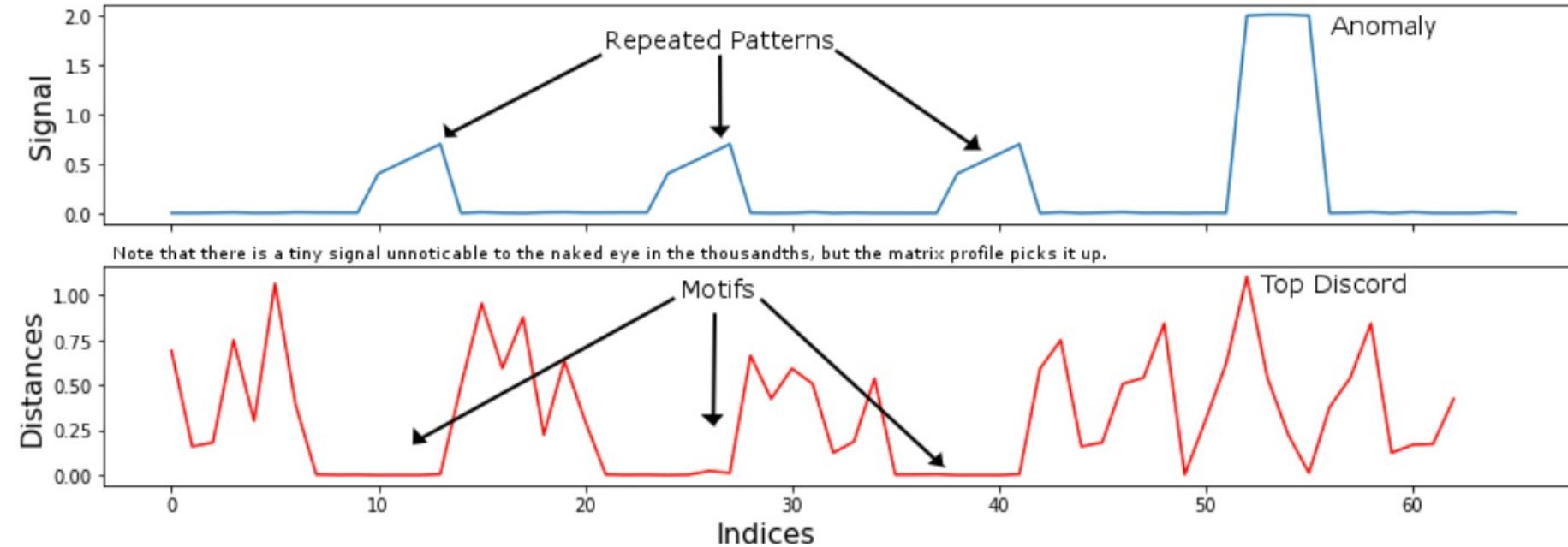
The general algorithm:

1. Compute the distances for the windowed sub-sequence against the entire time series
2. Set an exclusion zone to ignore trivial matches
3. Updates the distance profile with the minimal values
4. Set the first nearest-neighbor index



Matrix profiles: motifs and discords

When the matrix profile is computed, it is possible to identify **motifs (repeated patterns)** and **discords (anomalies)** in a time series.



<https://towardsdatascience.com/introduction-to-matrix-profiles-5568f3375d90>

Original paper: All Pairs Similarity Joins for Time Series: A Unifying View that Includes Motifs, Discords and Shapelets. Chin-Chia Michael Yeh, Yan Zhu, Liudmila Ulanova, Nurjahan Begum, Yifei Ding, Hoang Anh Dau, Diego Furtado Silva, Abdullah Mueen, Eamonn Keogh (2016). IEEE ICDM 2016.

Signature method

The **signature method** refers to a collection of feature extraction techniques for multivariate time series, derived from the theory of controlled differential equations.

→ Successfully applied in a wide range of ML tasks with sequential, e.g., chinese character recognition or feature extraction from financial data streams.

Given a multivariate time series $X: [a, b] \rightarrow \mathbb{R}^d$ (or more generally called **path** in this context), we define the increment of the i -th coordinate of the path as:

$$S(X)_{a,t}^i = \int_{a < s < t} dX_s^i = X_t^i - X_a^i$$

Signature method

For every $(i, j) \in \{1, \dots, d\}^2$ we define the double iterated integral:

$$S(X)_{a,t}^{i,j} = \int_{a < s < t} S(X)_{a,s}^i dX_s^j = \int_{a < r < s < t} dX_r^i dX_s^j$$

Similarly, we can define the **triple-iterated integral**:

$$S(X)_{a,t}^{i,j,k} = \int_{a < s < t} S(X)_{a,s}^{i,j} dX_s^k$$

And, recursively, we can construct the **k-fold iterated integral**:

$$S(X)_{a,t}^{i_1, \dots, i_k} = \int_{a < s < t} S(X)_{a,s}^{i_1, \dots, i_{k-1}} dX_s^{i_k}$$

Signature method

We define signature of a time series (or, path) the infinite series of all the iterated integrals, defined by:

$$S(X)_{a,b} = (1, S(X)_{a,b}^1, \dots, S(X)_{a,b}^d, S(X)_{a,b}^{1,1}, S(X)_{a,b}^{1,2}, \dots)$$

where the superscripts vary within the set of all multi-indexes

$$W = \{(i_1, \dots, i_k) | k \geq 1; i_1, \dots, i_k \in \{1, \dots, d\}\}$$



Data Mining with Time Series

Recap



In this lecture

- Introduction to Data Mining
- Frequency analysys
- Dynamic time warping
- Feature extraction techniques

Python resources

- **Tsfresh:** python library which is used to extract characteristics from time series.

<https://tsfresh.readthedocs.io/en/latest/>

Python resources

- **Tsfresh** is a Python library which is used to extract characteristics from time series.

<https://tsfresh.readthedocs.io/en/latest/>

- **TSFEL** is a Python package for feature extraction on time series data.

<https://tsfel.readthedocs.io/en/latest/>

Python resources

- **Tsfresh** is a Python library which is used to extract characteristics from time series.

<https://tsfresh.readthedocs.io/en/latest/>

- **TSFEL** is a Python package for feature extraction on time series data.

<https://tsfel.readthedocs.io/en/latest/>

- A systematic review of Python packages for time series, Siebert et al.

Paper: <https://arxiv.org/abs/2104.07406>

Website: <https://siebert-julien.github.io/time-series-analysis-python/>

Name	Tasks							Data Preparation				
	forecasting	classification	clustering	anomaly detection	segmentation	pattern recognition	change point detection	dimensionality reduction	missing values imputation	decomposition	preprocessing	simi mea
	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
arch	true	false	false	false	false	false	false	false	false	false	false	false
atspy	true	false	false	false	false	false	false	false	true	true	true	false
banpei	false	false	false	true	false	false	true	false	false	false	false	false
cesium	false	false	false	false	false	false	false	false	false	false	true	false
darts	true	false	false	false	false	false	false	false	true	true	true	false
deeptime	true	false	true	false	false	false	false	true	false	true	true	false
delタpy	true	false	true	false	false	false	false	true	false	true	true	true
dtaidistance	false	false	true	false	false	false	false	false	false	false	false	true
EMD-signal	false	false	false	false	false	false	false	false	true	false	false	false
flood-forecast	true	false	false	false	false	false	false	false	true	false	true	false



Machine Learning for Time Series

(MLTS or MLTS-Deluxe Lectures)

Dr. Dario Zanca

Machine Learning and Data Analytics (MaD) Lab
Friedrich-Alexander-Universität Erlangen-Nürnberg
17.01.2023

- Time series fundamentals and definitions (2 lectures)
- Bayesian Inference (1 lecture)
- Gaussian processes (2 lectures)
- State space models (2 lectures)
- Autoregressive models (1 lecture)
- Data mining on time series (1 lecture)
- Deep learning on time series (4 lectures) ←
- Domain adaptation (1 lecture)

In this lecture...

- 1. Introduction to Deep Learning (DL)**
- 2. Recurrent Neural Networks (RNNs)**
- 3. Backpropagation Through Time (BPTT)**

Why deep learning?

Previous method needed **handcrafted features**:

- MFCCs (speech processing) (1)
- I-Vector (speech processing) (2)
- Sift (scene alignment, videos) (3) → **Needs expert knowledge about domain**

(1) Mermelstein, P. (1976). Distance measures for speech recognition, psychological and instrumental. *Pattern recognition and artificial intelligence*, 116, 374-388.

(2) V. Gupta, P. Kenny, P. Ouellet and T. Stafylakis, "I-vector-based speaker adaptation of deep neural networks for French broadcast audio transcription," *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2014, pp. 6334-6338, doi: 10.1109/ICASSP.2014.6854823.

(3) Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2), 91-110.

Why deep learning?

Previous method needed **handcrafted features**:

- MFCCs (speech processing) (1)
- I-Vector (speech processing) (2)
- Sift (scene alignment, videos) (3) → **Needs expert knowledge about domain**

What if we can not define generally applicable features?

- High dimensional data
- Hard to come up with generally applicable feature
 - With DL we can find features in a data driven way (e.g., Yolo beats prior approaches (4))

(4) Dean, T., Ruzon, M. A., Segal, M., Shlens, J., Vijayanarasimhan, S., & Yagnik, J. (2013). Fast, accurate detection of 100,000 object classes on a single machine. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 1814-1821).



Deep learning for Time Series – Recurrent models

Introduction to Deep Learning



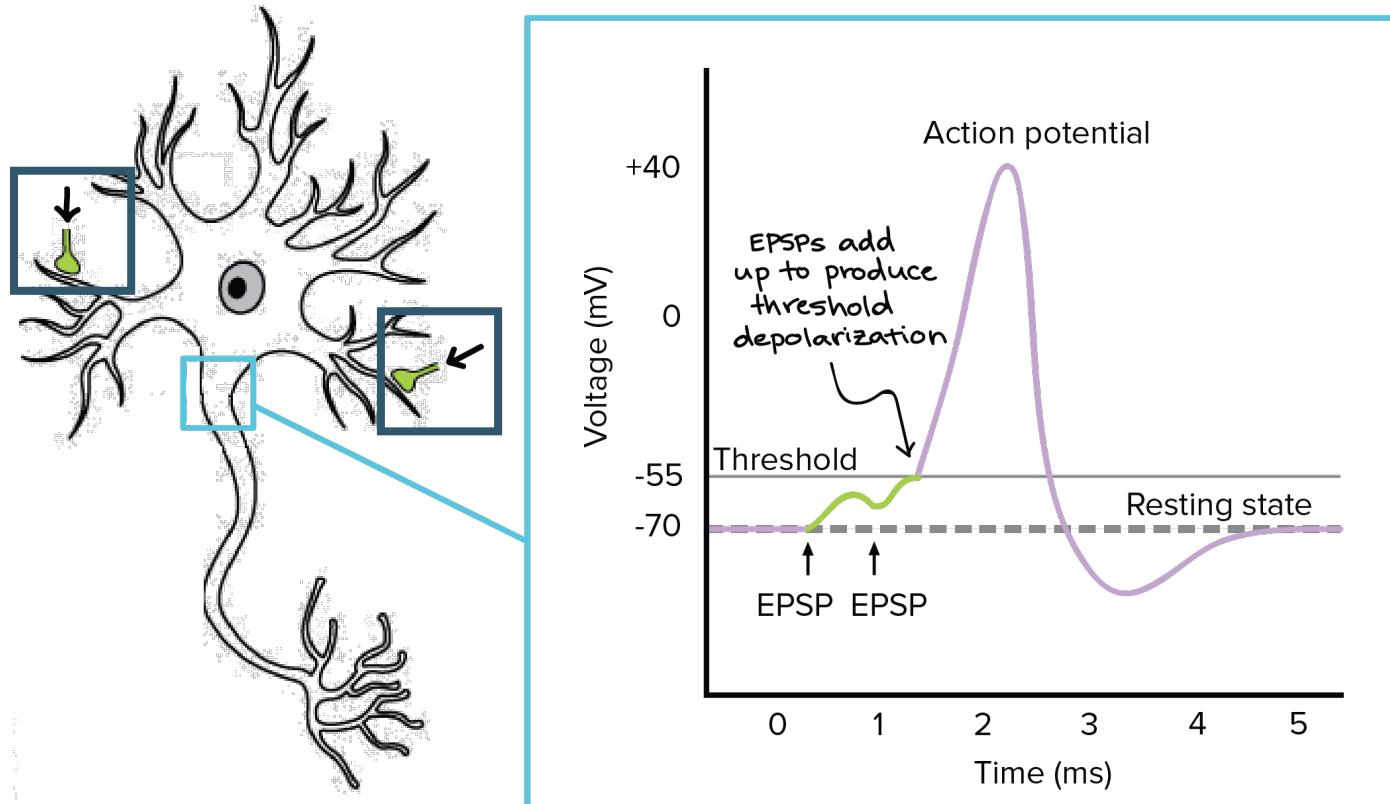
The Human Brain

The human brain is our reference for an intelligent agent, that

- contains different areas specialized for some tasks (e.g., the visual cortex)
- consists of neurons as the fundamental unit of “computation”



The Brain's Neuron



- Excitatory **stimuli** reach the neuron
- Threshold is reached
- Neuron fires and triggers **action potential**

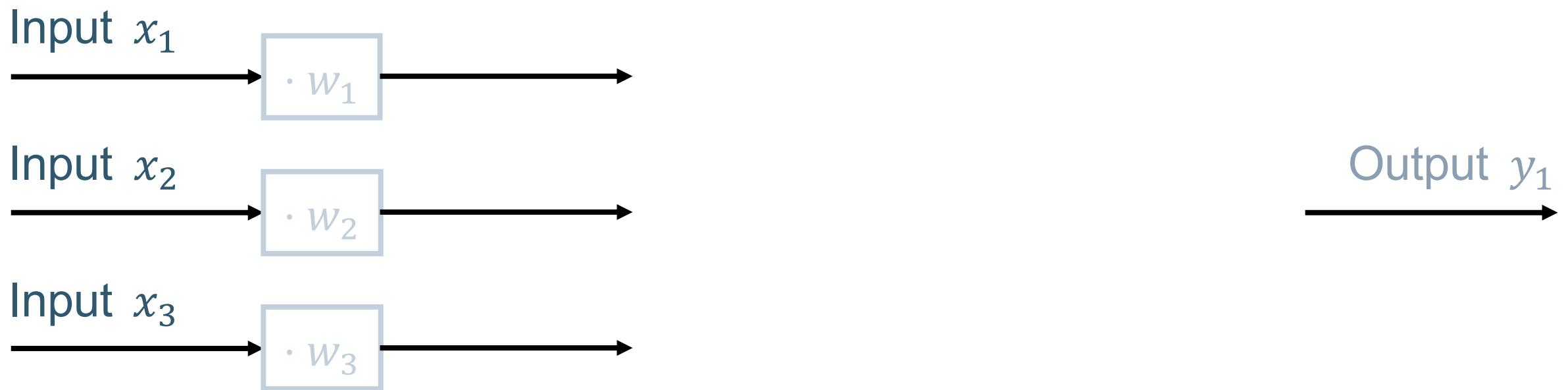
The Perceptron – Computational model of a neuron

1. Let's start by adding some basic components (**input** and **output**), we're subsequently going to build the computational model step by step



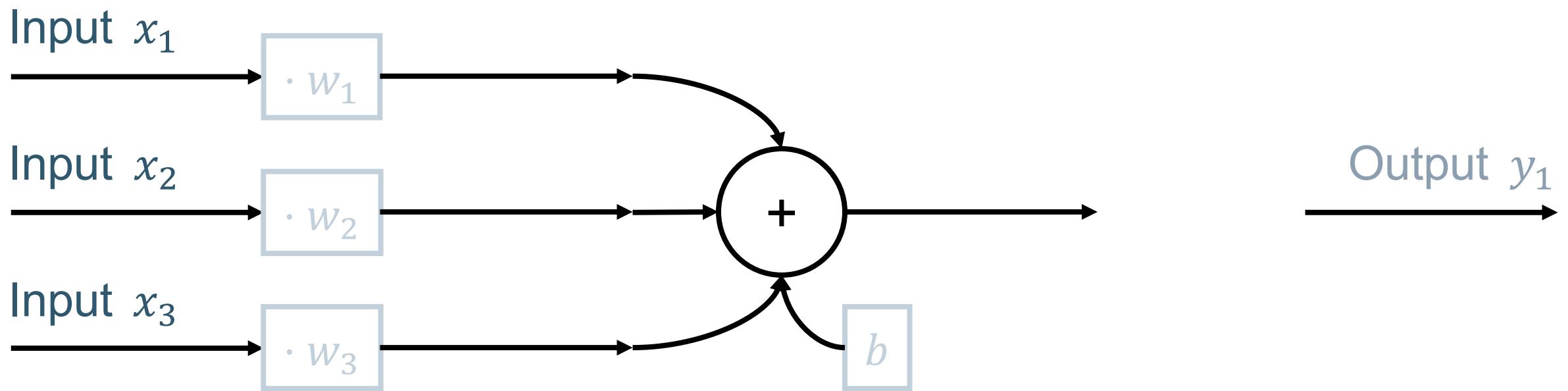
The Perceptron – Computational model of a neuron

- Weights can “select” or “deselect” input channels (not all are relevant for subsequent computations)



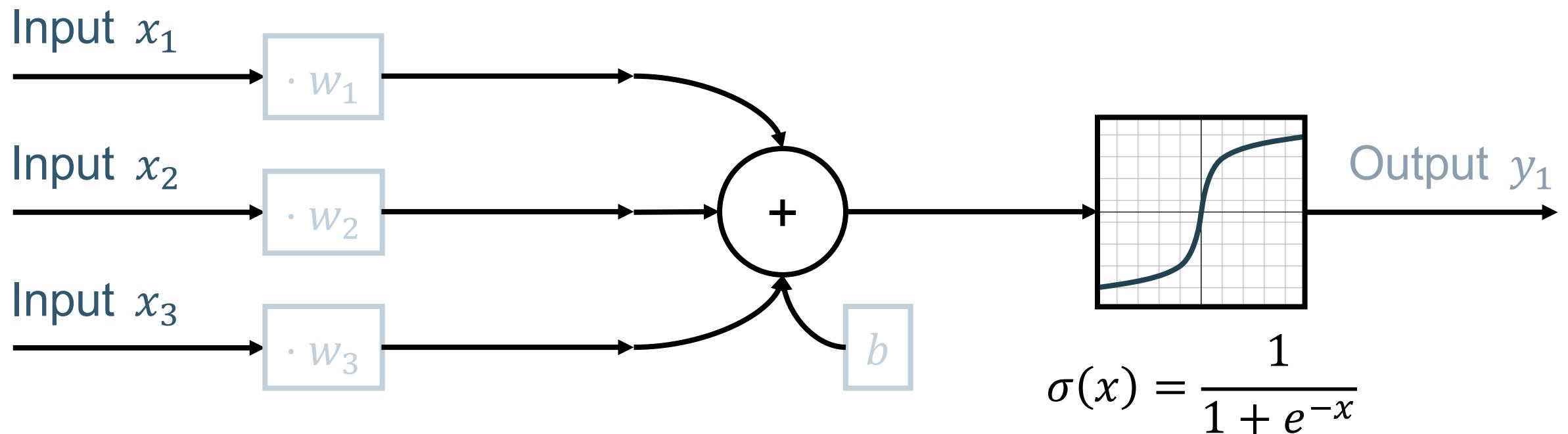
The Perceptron – Computational model of a neuron

3. We add up all the excitatory signals and the resting potential to determine the current potential.



The Perceptron – Computational model of a neuron

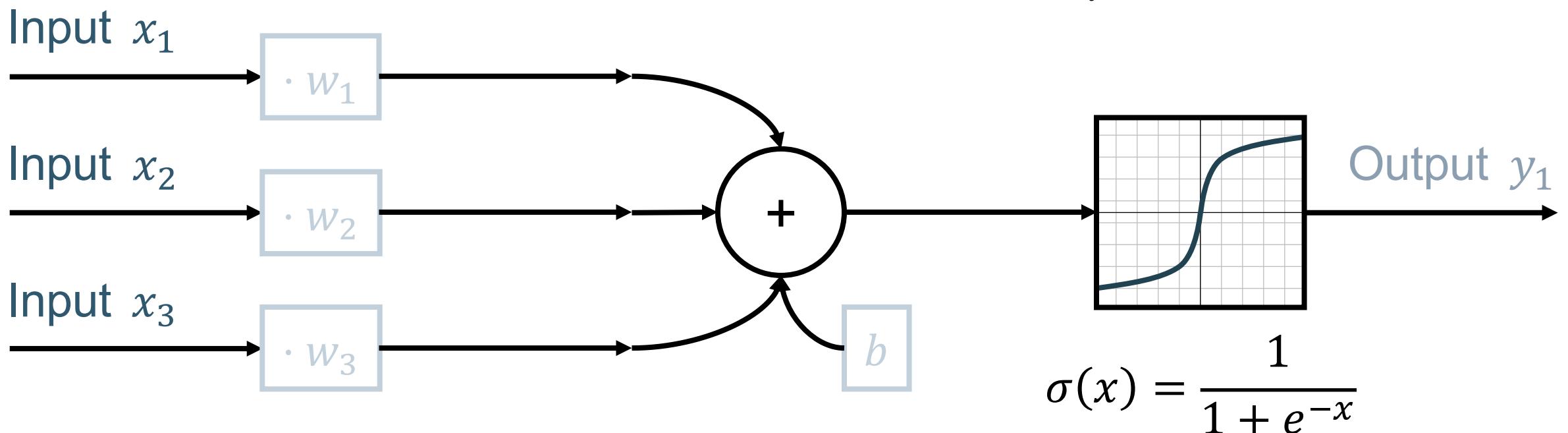
4. A **threshold function σ** is applied to determine whether an action potential has to be sent in the **output**



The Perceptron – Computational model of a neuron

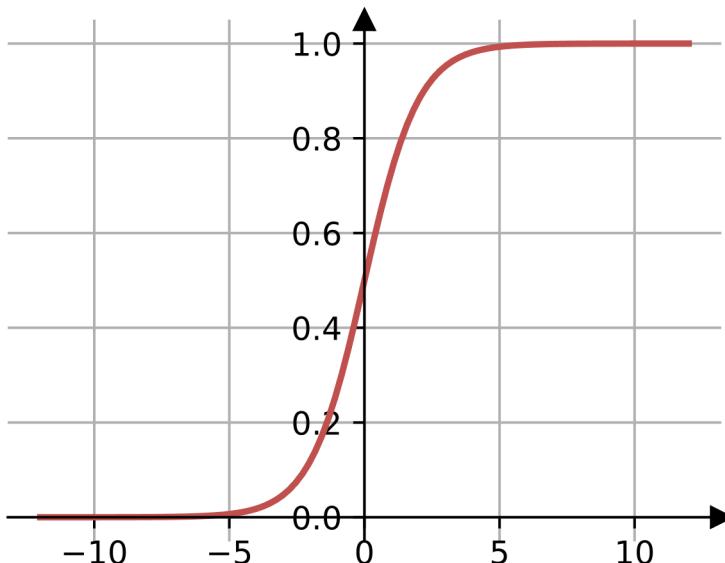
5. We can write the perceptron mathematical model to map inputs x_1, x_2, x_3 to the output y_1 using channel weights w_1, w_2, w_3, b :

$$y_1 = \sigma(w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_3 + b) = \sigma\left(\sum_i w_i \cdot x_i + b\right)$$



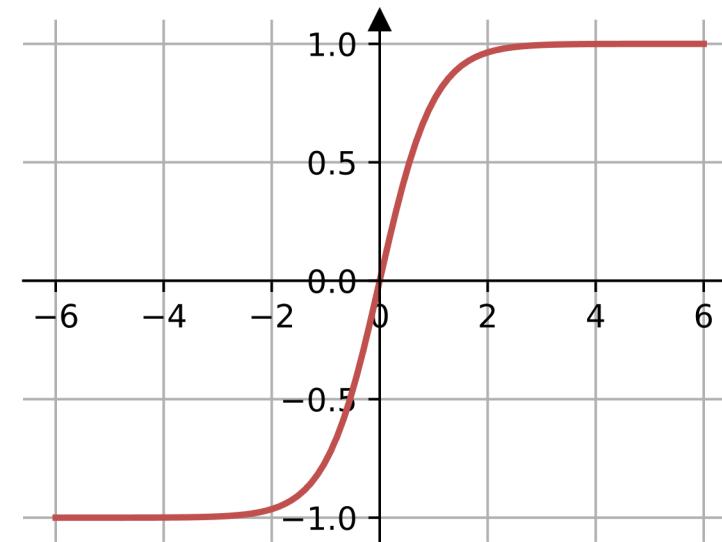
Activation functions

Sigmoid



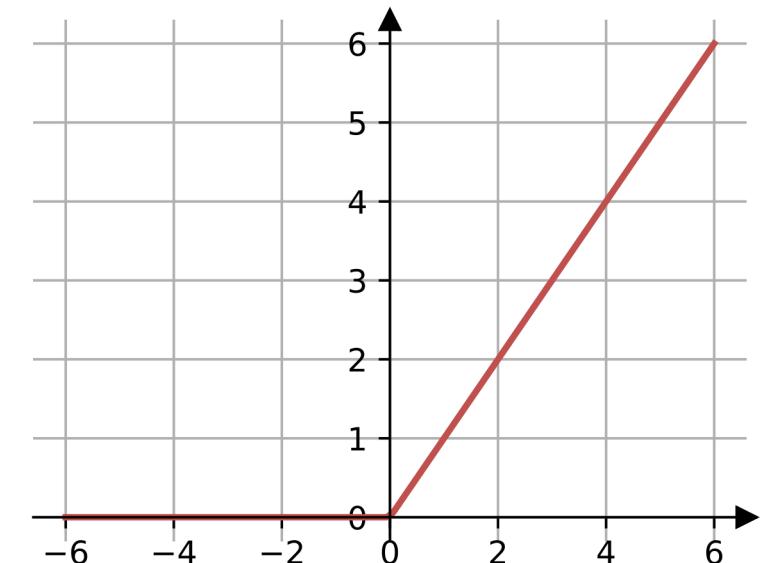
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Hyperbolic Tangent



$$\sigma(x) = \tanh(x)$$

Rectified Linear Unit



$$\sigma(x) = \max(x, 0)$$

Shallow networks

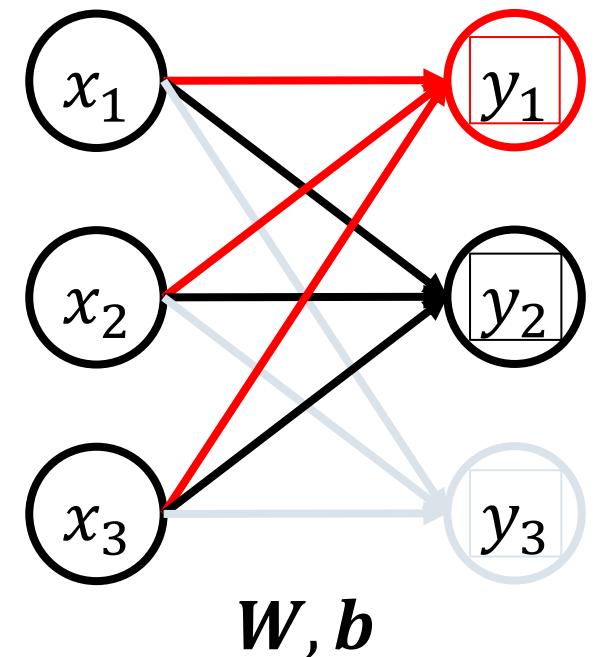
We can combine multiple perceptrons to create a **layer**.

We can thus rewrite the three computations as:

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

Or in a more simplified form:

$$y = \sigma(W \cdot x + b)$$



Multilayer perceptron (MLP)

We can chain multiple layers, with each output being the input of the next:

$$y = \sigma(W^1 \cdot x + b^1)$$

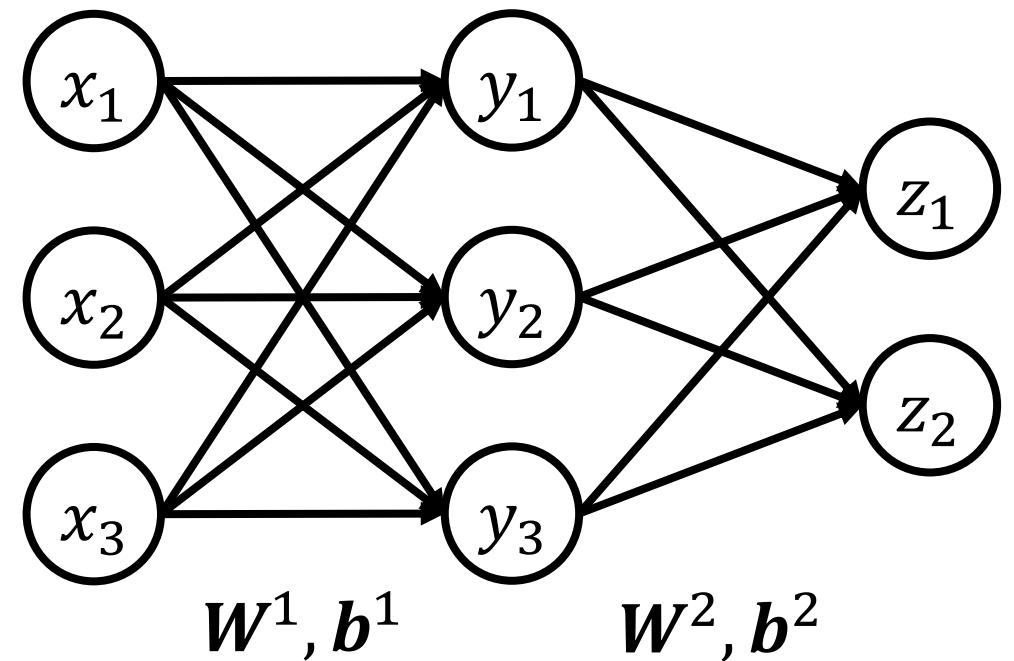
$$z = \sigma(W^2 \cdot y + b^2)$$

Combining these leads us to:

$$z = \sigma(W^2 \cdot \sigma(W^1 \cdot x + b^1) + b^2)$$

- Each layer has its own set of parameters (weights W^i and bias b^i)
- The underlying computation is a matrix multiplication described by

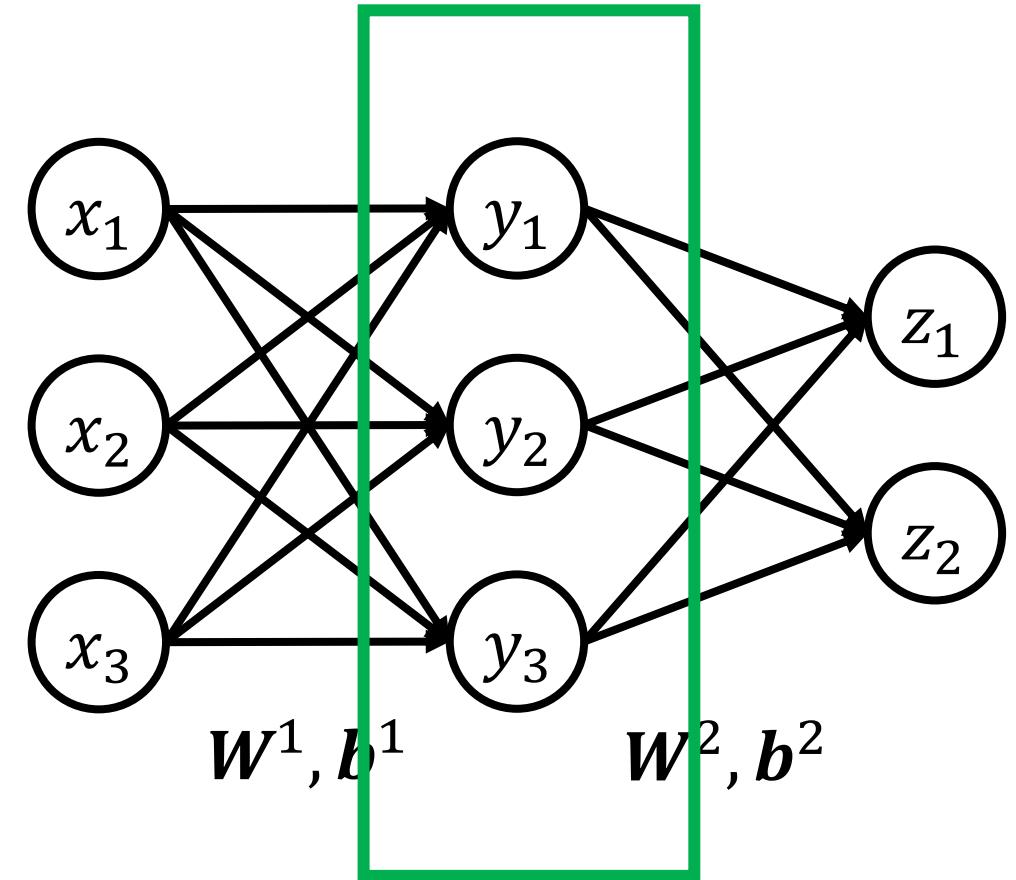
$$\mathbf{y}^{i+1} = \sigma(\mathbf{W}^i \cdot \mathbf{y}^i + \mathbf{b}^i)$$



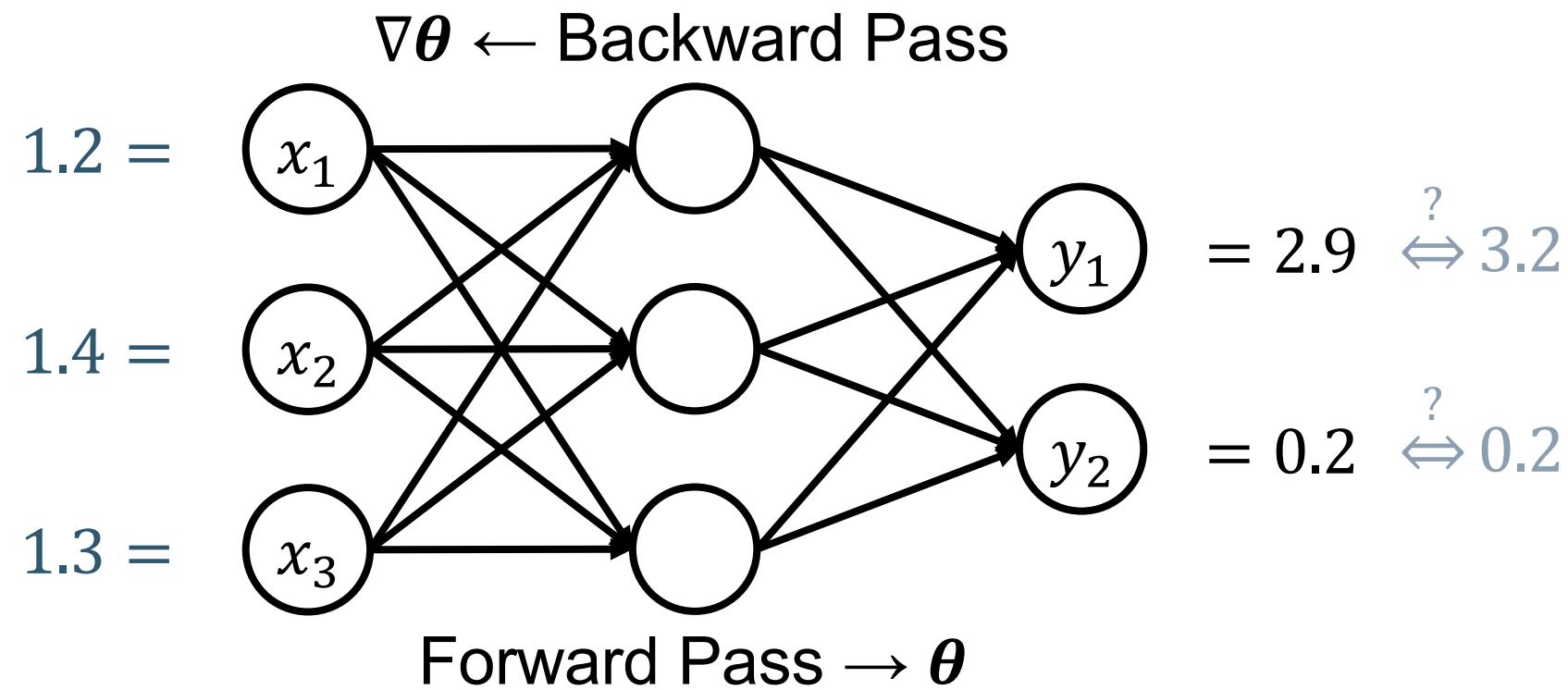
Multilayer perceptron (MLP)

We call “**hidden layer**” any layer in between the input and the output layers.

- For example: the neural network (image on the right) is a Multi-Layer-Perceptron (MLP) with a single hidden layer (highlighted by the green box).



How do we learn model parameters?



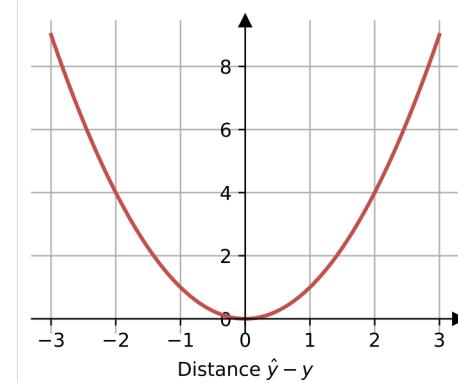
The loss function

The loss function is a comparison metric between the predicted outputs \hat{y}_i and the expected outputs y_i .

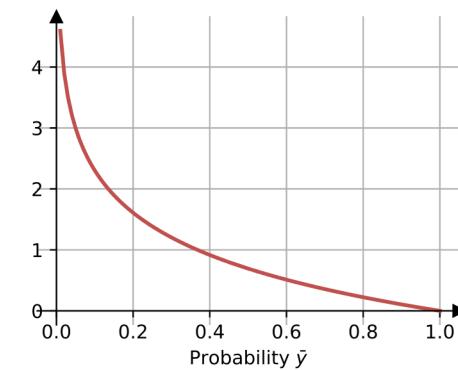
The choice of the loss function usually depends on the type of problem:

- For regression, a common metric is the mean squared error
- For classification, a common metric is the cross entropy

$$MSE(\hat{y}, y) = \frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i)^2$$



$$CE(\hat{y}, y) = - \sum_i y_i \log(\hat{y}_j)$$



Gradient descent

Gradient Descent can be used to incrementally adjusts the parameters θ based on the gradient $\nabla\theta$ of the parameters

- For each iteration:
 - Compute the error of the parameters θ
 - Compute the gradient of the parameters $\nabla\theta$
 - Update parameters using $\theta^{i+1} = \theta^i - \lambda \cdot \nabla\theta^i$

where we denoted $\nabla\theta^i = \partial\mathcal{L}/\partial\theta^i$.

- The learning rate λ can lead to slow convergence if not properly configured
- There is no guarantee that the algorithm converges to the global optimum (e.g., due to learning rate λ or initial parameters θ^1)

Backpropagation (BP)

Backpropagation algorithm is widely used to train artificial neural networks.

- In *fitting* a neural network, backpropagation computes **efficiently** the gradients of the loss function with respect to all weights in the network.
- This efficiency makes it feasible to use **gradient methods** for training multilayer networks, updating weights to minimize loss, like gradient descent.
- BP algorithm makes use of the **chain rule**, computing the gradient one layer at a time, to avoid redundant calculations.

Backpropagation (BP)

Let a_i^k be the activation of the i -th neuron in the k -th layer. This is related to the previous $(k - 1)$ -th layer by

$$a_i^k = \sum_{j=0}^{r_{k-1}} w_{ji}^k o_j^{k-1}$$

where r_{k-1} is the number of units in the $(k - 1)$ -th layer, and we simplified the notation incorporating the bias in the weights vector, as w_{0i}^k element.

Backpropagation (BP)

Let the loss function be:

$$E(X, \theta) = \frac{1}{2n} \sum_{i=1}^n (\hat{y}_i - y_i)^2$$

where y is the desired output and \hat{y}_i is the predicted output from the neural network.

Backpropagation (BP)

The derivation of the backpropagation algorithm begins by applying the chain rule to the error function partial derivative

$$\frac{\partial E}{\partial w_{ij}^k} = \frac{\partial E}{\partial a_j^k} \frac{\partial a_j^k}{\partial w_{ij}^k}$$

where the first term is usually called error and denoted by $\delta_j^k = \frac{\partial E}{\partial a_j^k}$

and the second term $\frac{\partial a_j^k}{\partial w_{ij}^k} = \frac{\partial}{\partial w_{ij}^k} \left(\sum_{j=0}^{r_{k-1}} w_{ji}^k o_j^{k-1} + b_j^l \right) = o_i^{k-1}$.

Thus, $\frac{\partial E}{\partial w_{ij}^k} = \delta_j^k o_i^{k-1}$

Backpropagation (BP)

Now, considering the final layer m , let σ be the activation function of the final layer, applying the partial derivatives and using the chain rule we obtain

$$\delta_j^m = (\hat{y} - y)\sigma'(a_j^m)$$

which we can exploit to compute the error w.r.t. a specific weight:

$$\frac{\partial E}{\partial w_{ij}^m} = \delta_j^m o_i^{m-1} = (\hat{y} - y)\sigma'(a_j^m)o_i^{m-1}$$

Backpropagation (BP)

For an hidden layer k , the error term is given by

$$\delta_j^k = \sigma'(a_j^k) \sum_{i=1}^{r_{k+1}} w_{ji}^{k+1} \delta_i^{k+1}$$

which, similarly, we can exploit to compute the error w.r.t. a specific weight:

$$\frac{\partial E}{\partial w_{ij}^k} = \delta_j^k o_i^{k-1} = \sigma'(a_j^k) \left(\sum_{l=1}^{r_{k+1}} w_{jl}^{k+1} \delta_l^{k+1} \right) o_i^{k-1}$$

Backpropagation (BP)

Finally the update for each single weight is given by

$$\Delta w_{ij}^k = -\lambda \frac{\partial E}{\partial w_{ij}^k}$$

Backpropagation (BP): algorithmic view

1. Calculate the forward pass and store results for \hat{y} , a_j^k , and o_j^k .
2. Calculate the backward pass and store results for $\frac{\partial E}{\partial w_{ij}^k}$, proceeding from the last layer:
 - a) Evaluate the error terms for the last layer δ_j^m
 - b) Backpropagate the error term for the computation of δ_j^k
 - c) Proceed to all previous layers
3. Combine the individual gradients $\forall j$ (simple average)
4. Update the weights according to a learning rate λ



Deep learning for Time Series – Recurrent models

Recurrent Neural Networks (RNNs)

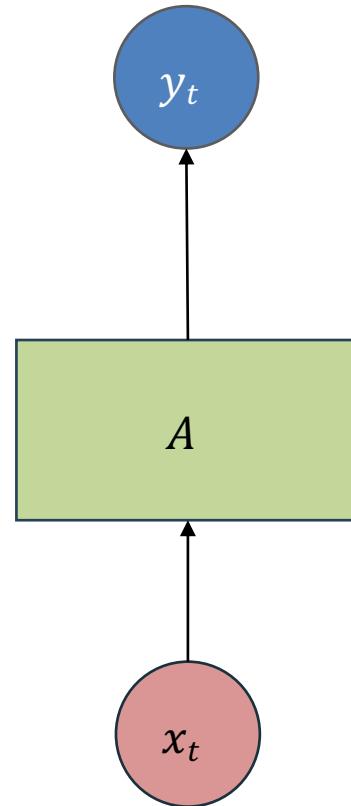


Limitations of NN for time series data

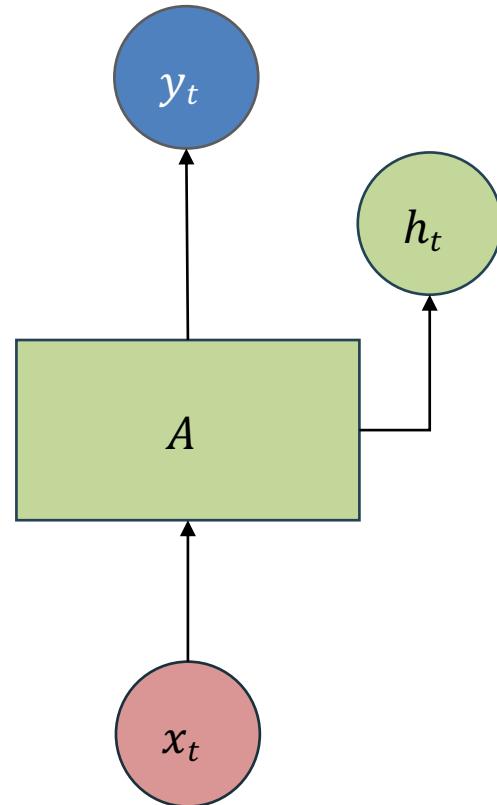
Feed-forward neural networks present some **disadvantages**, when applied to sequential data:

- Cannot work online (sequence has to be fed all at once)
- Consider only the current input
- Cannot memorize previous time steps
- Inefficient
- Cannot handle directly sequences of different lengths

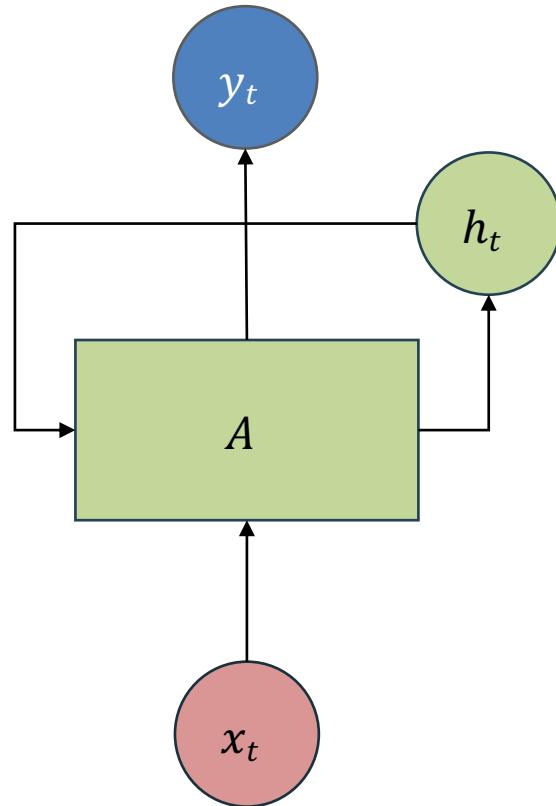
Recurrent Neural Network (RNN)



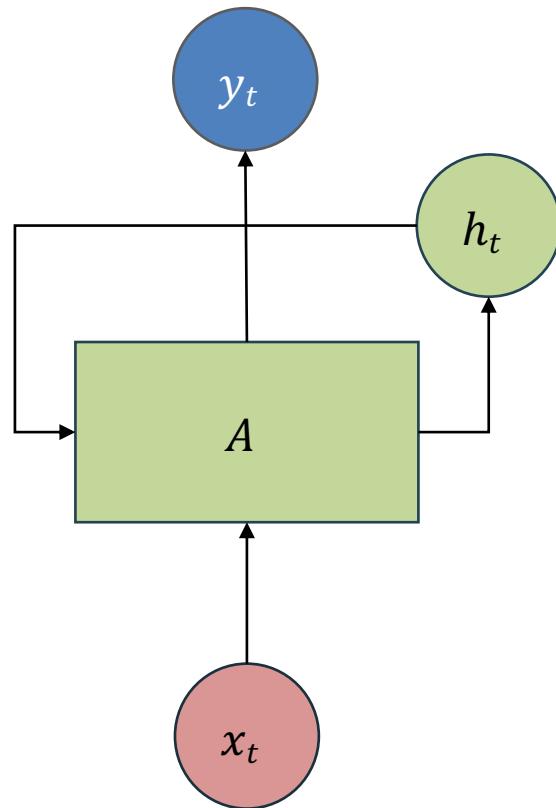
Recurrent Neural Network (RNN)



Recurrent Neural Network (RNN)



Recurrent Neural Network (RNN)



A **recurrent neural network (RNN)** is a neural network that contains feed-back connections.

- Activations can flow in a loop
- It allows for temporal processing

An RNN is composed by:

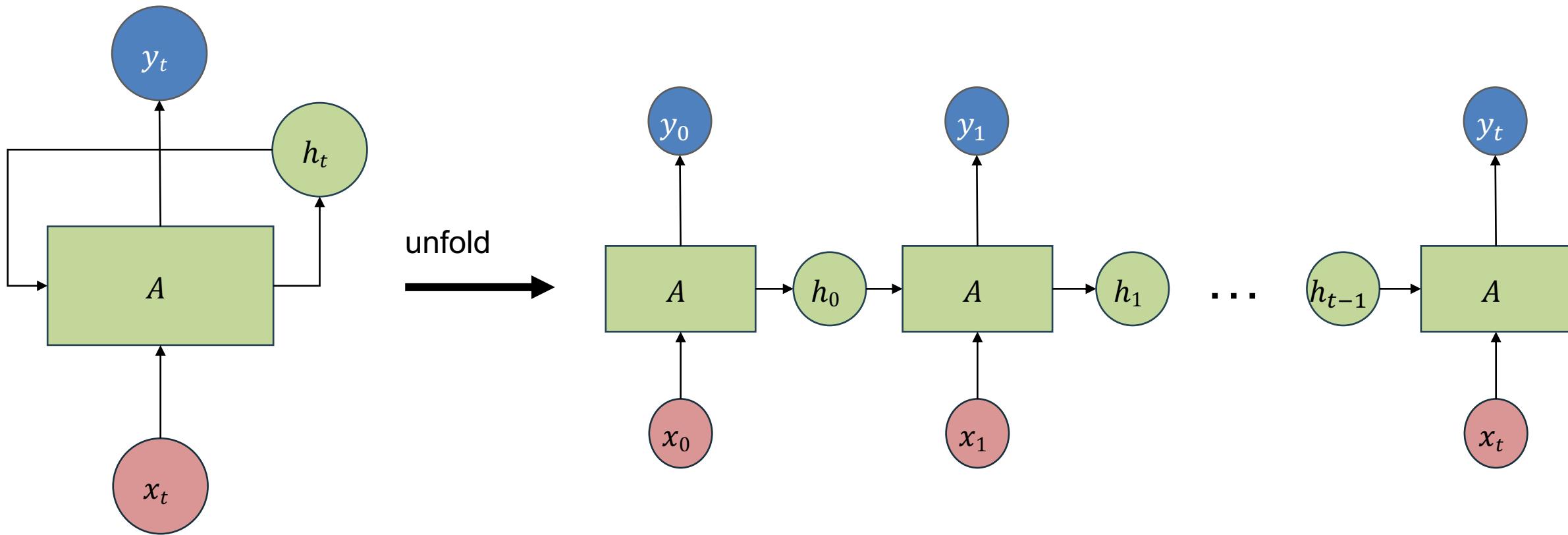
x_t : input at time t

A : neural network

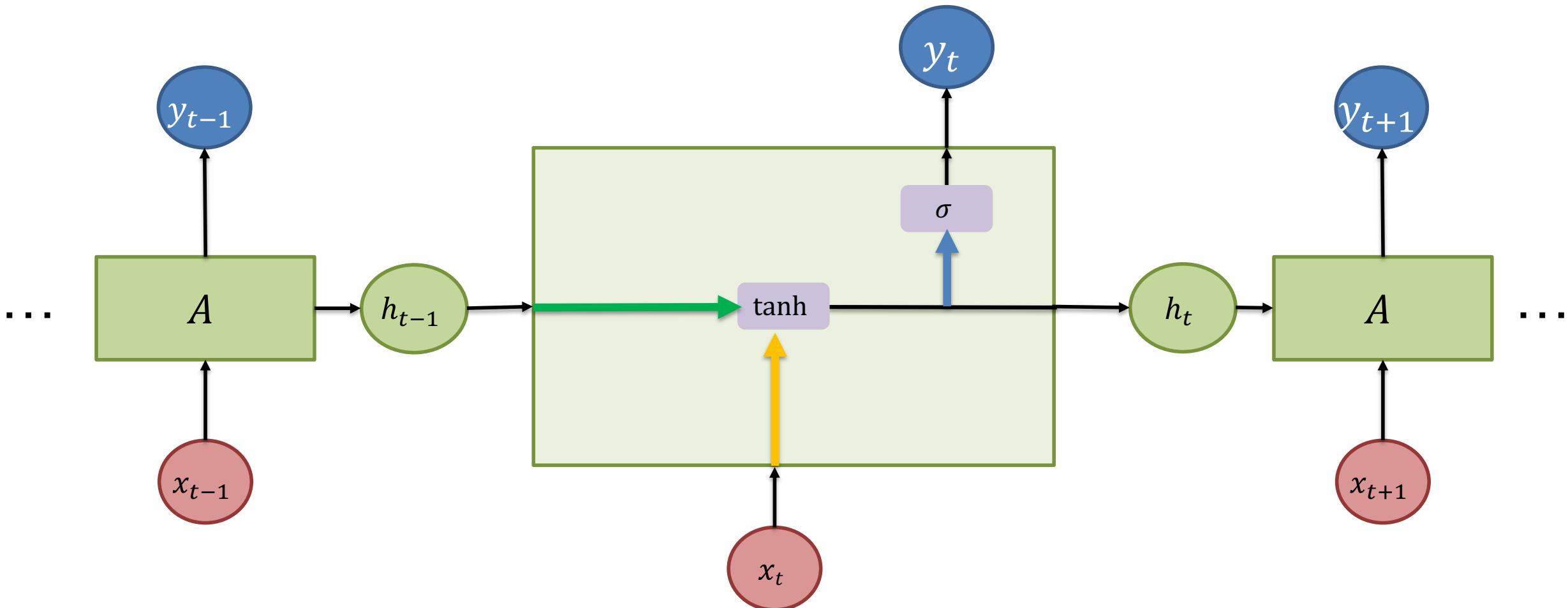
h_t : hidden state

y_t : output at time t

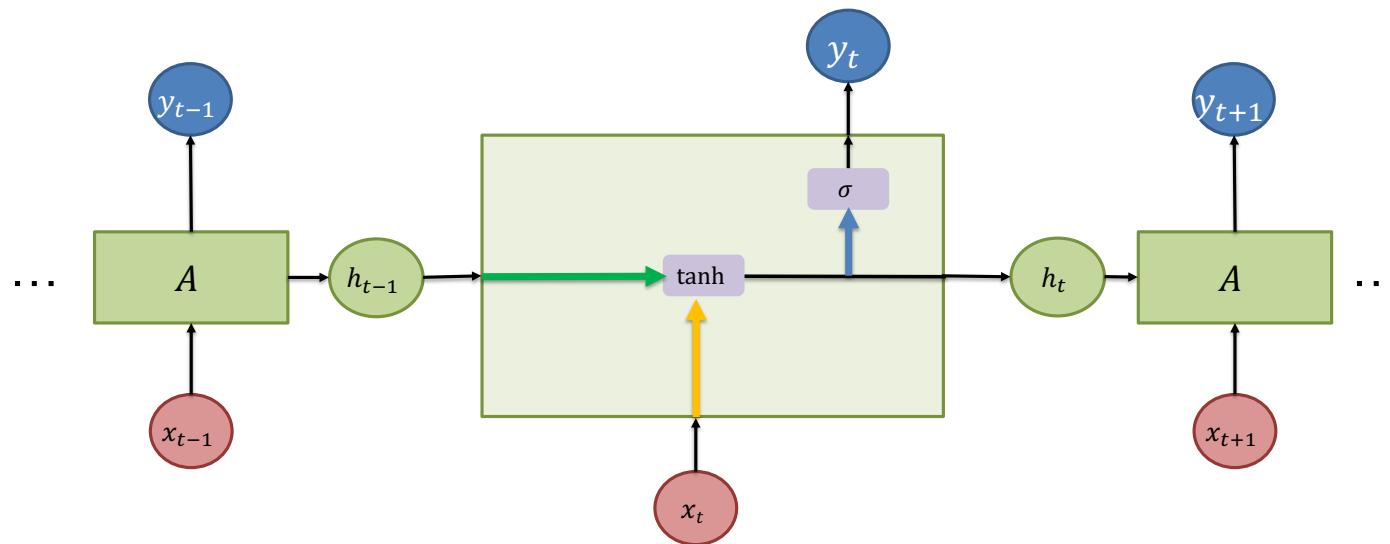
RNN Unfolding



Mathematical formulation



Mathematical formulation



The behaviour of the RNN can be described as a **dynamical system** by the pair of non-linear matrix equations:

$$h_t = \tanh(\mathbf{W}_{hh}h_{t-1} + \mathbf{W}_{xh}x_t)$$

$$y_t = \sigma(\mathbf{W}_{hy}h_t)$$

The order of the dynamical system corresponds to the dimensionality of the state h_t .

Mathematical formulation

In general, the neural network A can represent any function. We can write a more general formulation of the system as:

$$h_t = f_h(h_{t-1}, x_t; \theta_h)$$

$$y_t = f_y(h_t; \theta_y)$$

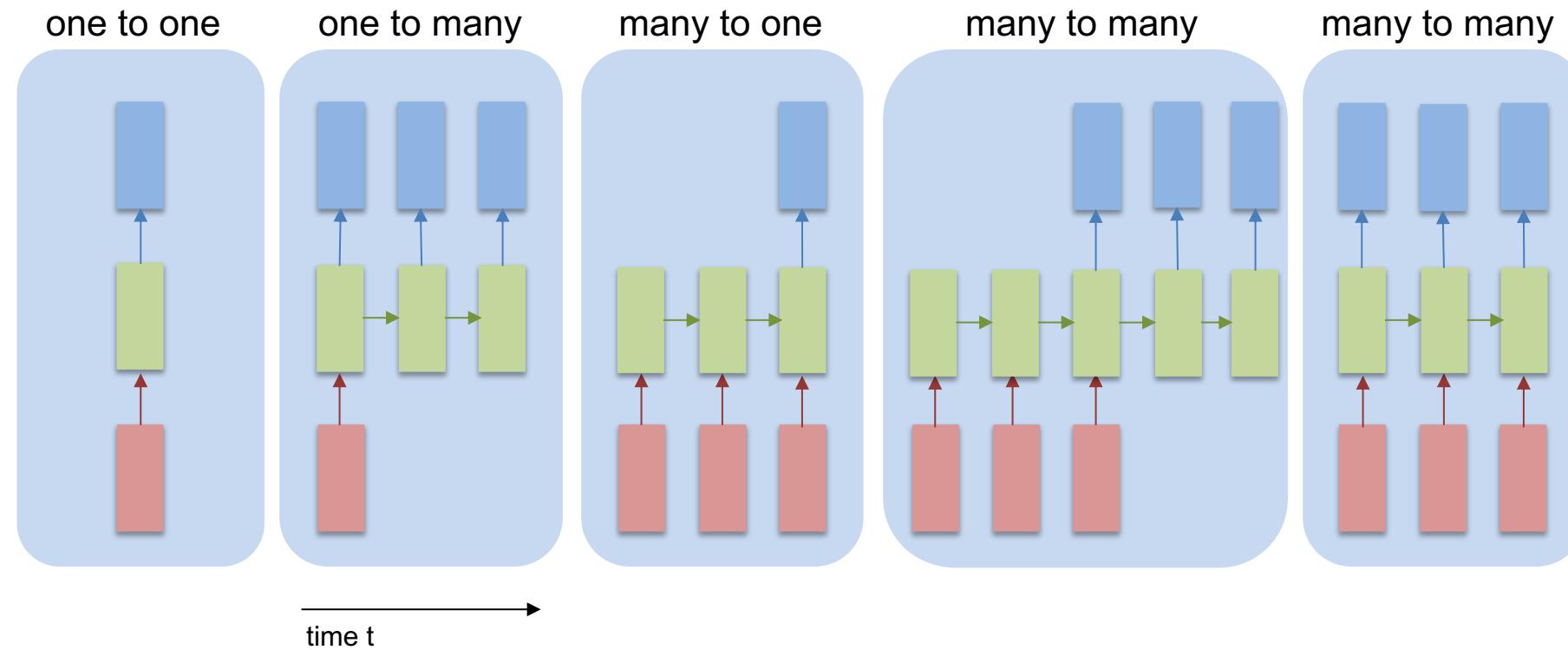
RNNs are universal approximators

The Universal Approximation Theorem states that:

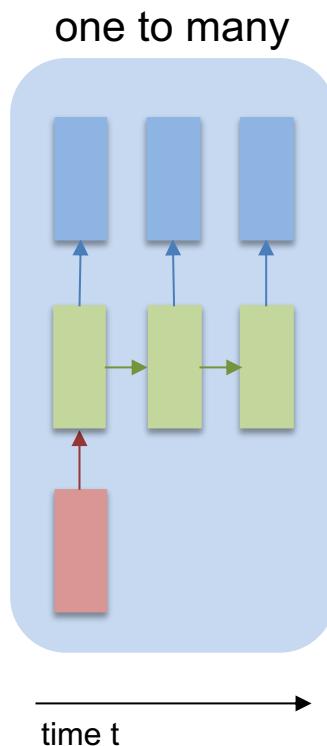
„Any non-linear dynamical system can be approximated to any accuracy by a recurrent neural network, with no restrictions on the compactness of the state space, provided that the network has enough sigmoidal hidden units.“

- Knowing that RNNs are universal approximators does not explain how to learn such dynamical system from data.
- Since we can think about RNNs in terms of dynamical systems, we can also investigate their properties like stability, controllability and observability.

RNNs architecture

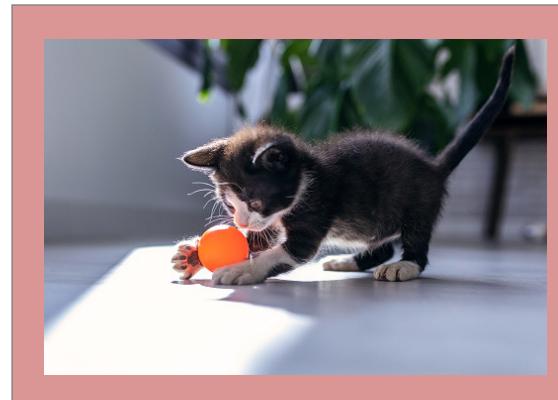


Example: one to many



A typical example of a one to many problem is that of **image captioning**.

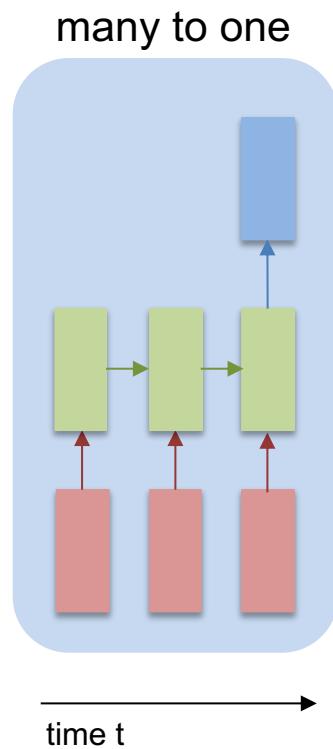
Input:



Output:

A cat playing with a ball

Example: many to one



A typical example of a many to one problem is that of **sentiment analysis**.

Input:

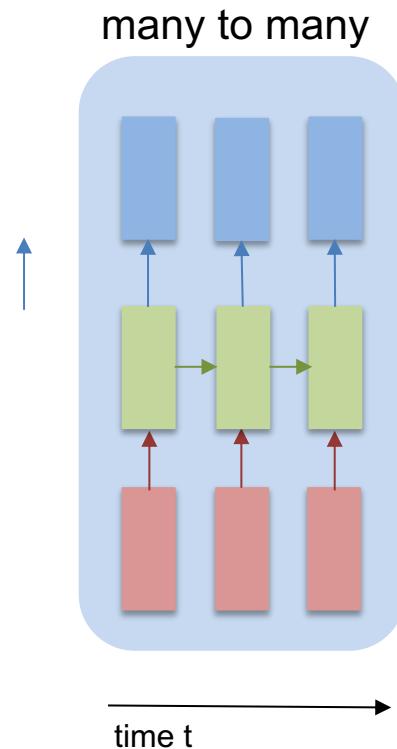
Horrible service the room was dirty

Output:



("negative")

Example: many to many



A typical example of a many to many problem is that of name entity recognition.

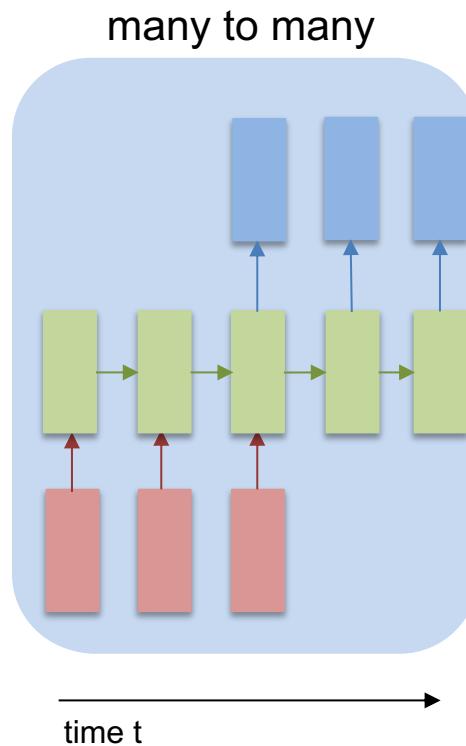
Input:

Harry Potter and Hermione invented a new spell

Output:

1 1 0 1 0 0 0 0

Example: many to many



Another example of a many to many problem is that of machine translation.

Input:

Horrible service the room was dirty

Output:

Un servizio orribile la camera era sporca



Deep learning for Time Series – Recurrent models

Backpropagation Through Time (BPTT)



Backpropagation Through Time (BPTT)

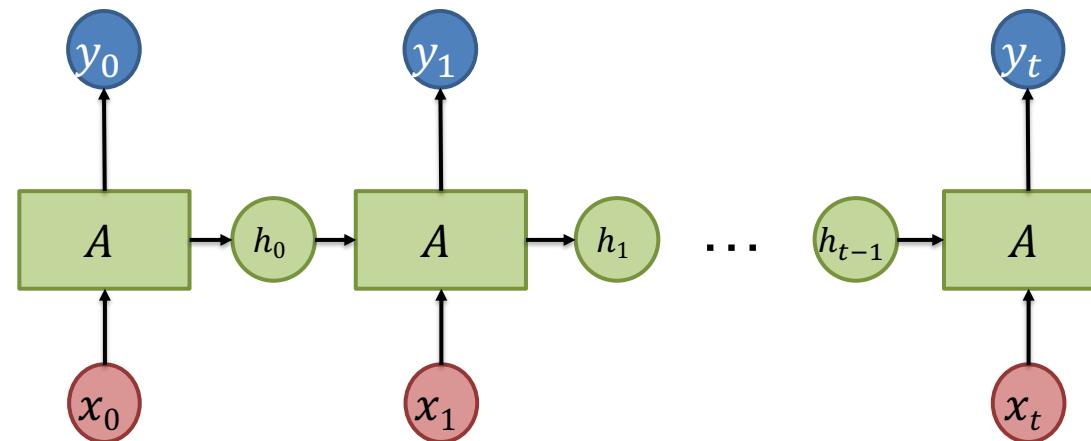
Backpropagation Through Time (BPTT) learning algorithm is a natural extension of the standard backpropagation that performs gradient descent on a complete unfolded RNN.

General idea:

1. Feed training examples through the network
2. Calculate the loss for the whole sequence
3. Get the gradients of all weights and update them according to the learning rate

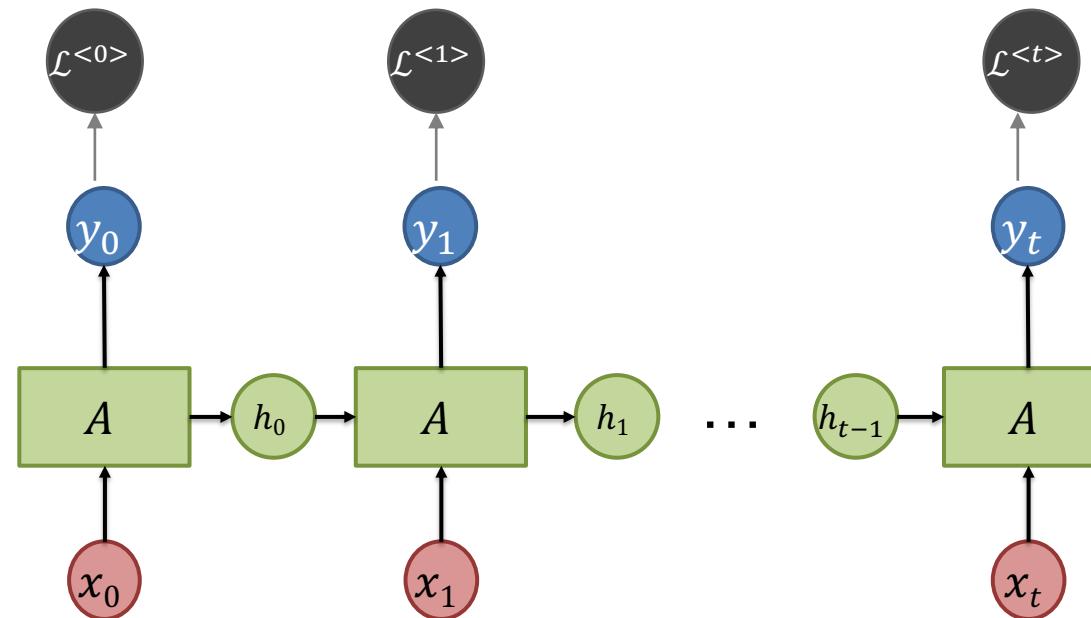
BPTT: Forward propagation

1. We feed the network with the whole sequence and compute all activations and outputs



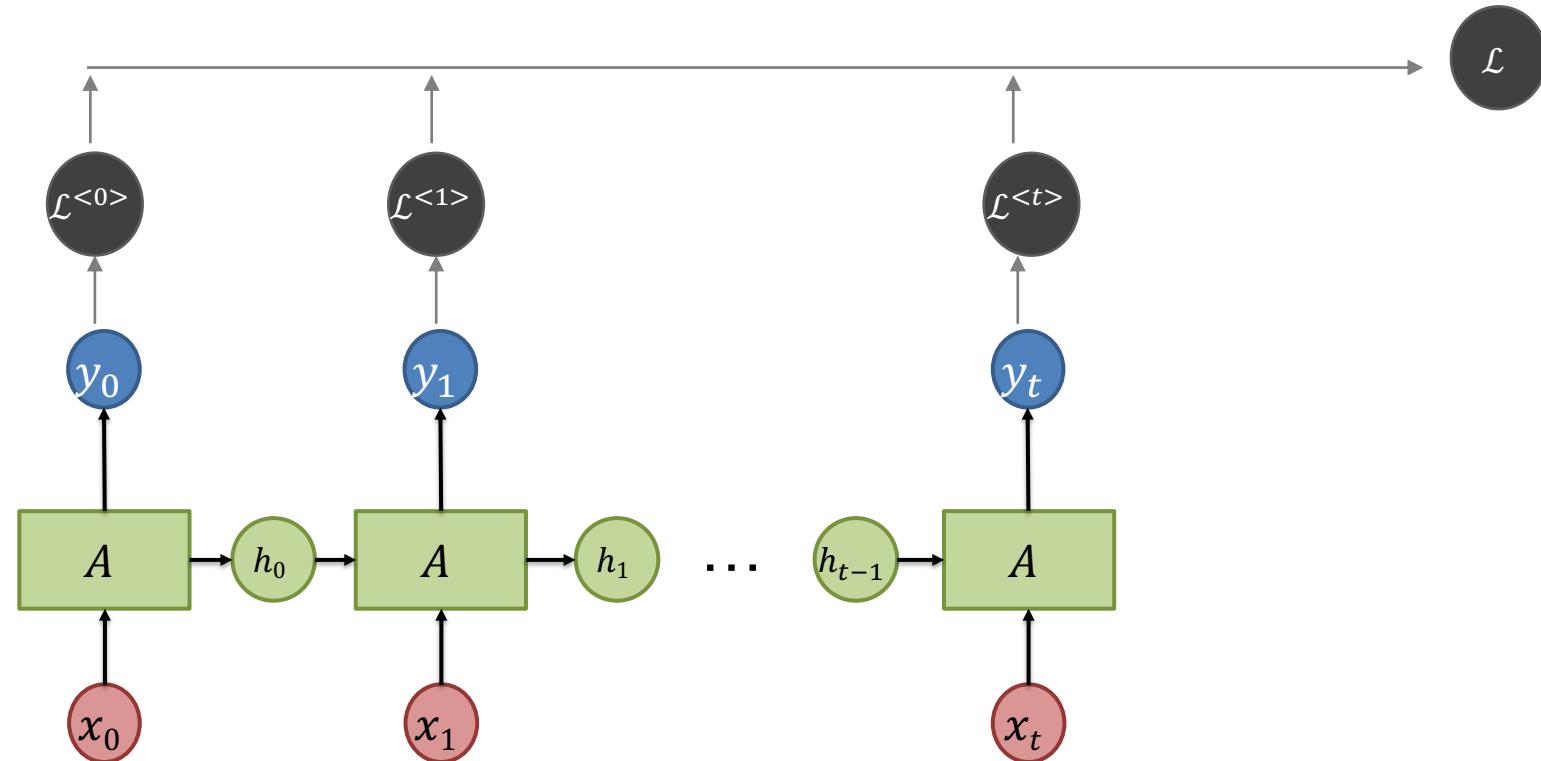
BPTT: Loss computation

2. We compute the overall loss of our prediction \hat{y} w.r.t. the true sequence y .



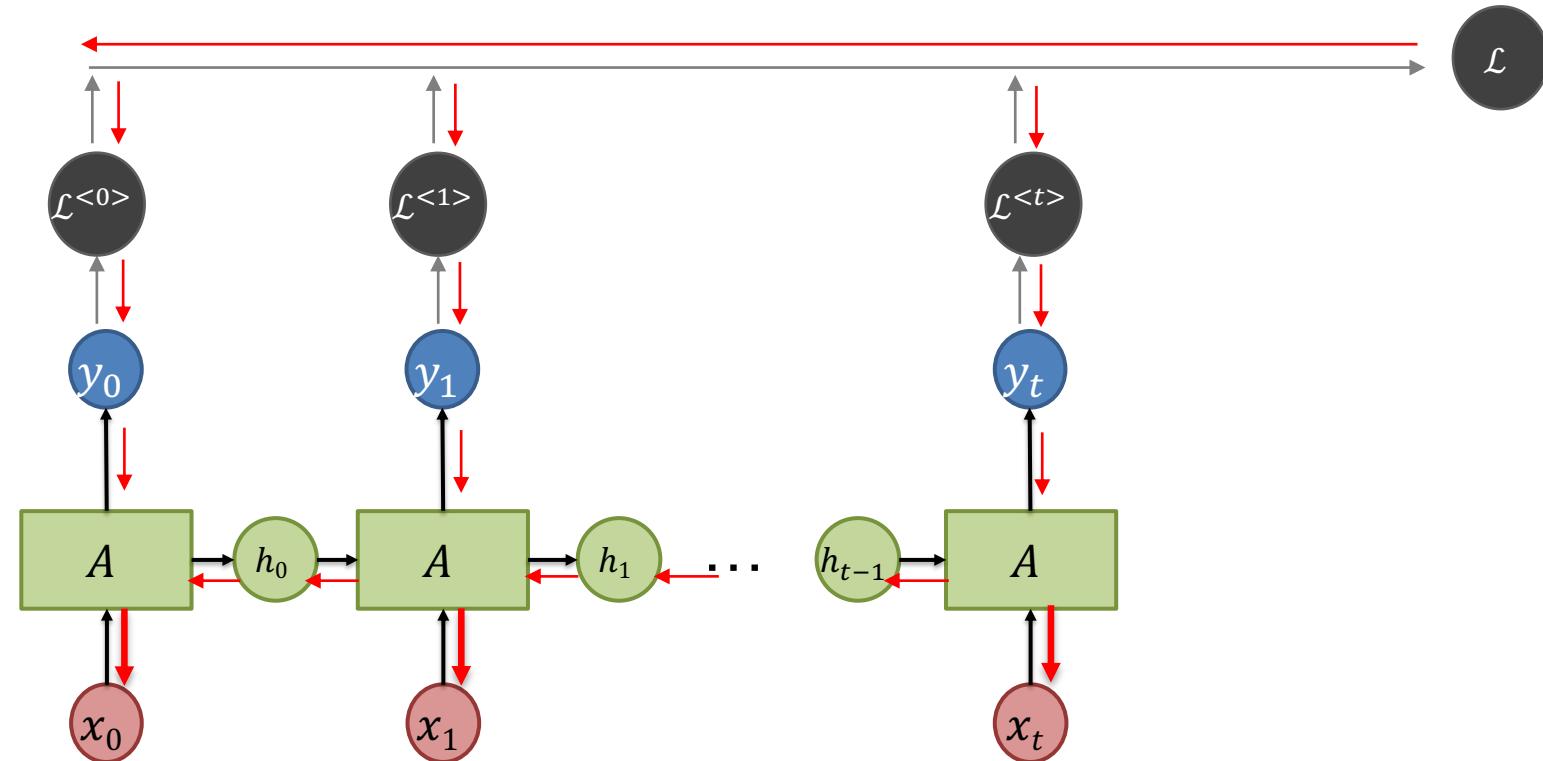
BPTT: Loss computation

2. We compute the overall loss of our prediction \hat{y} w.r.t. the true sequence y .



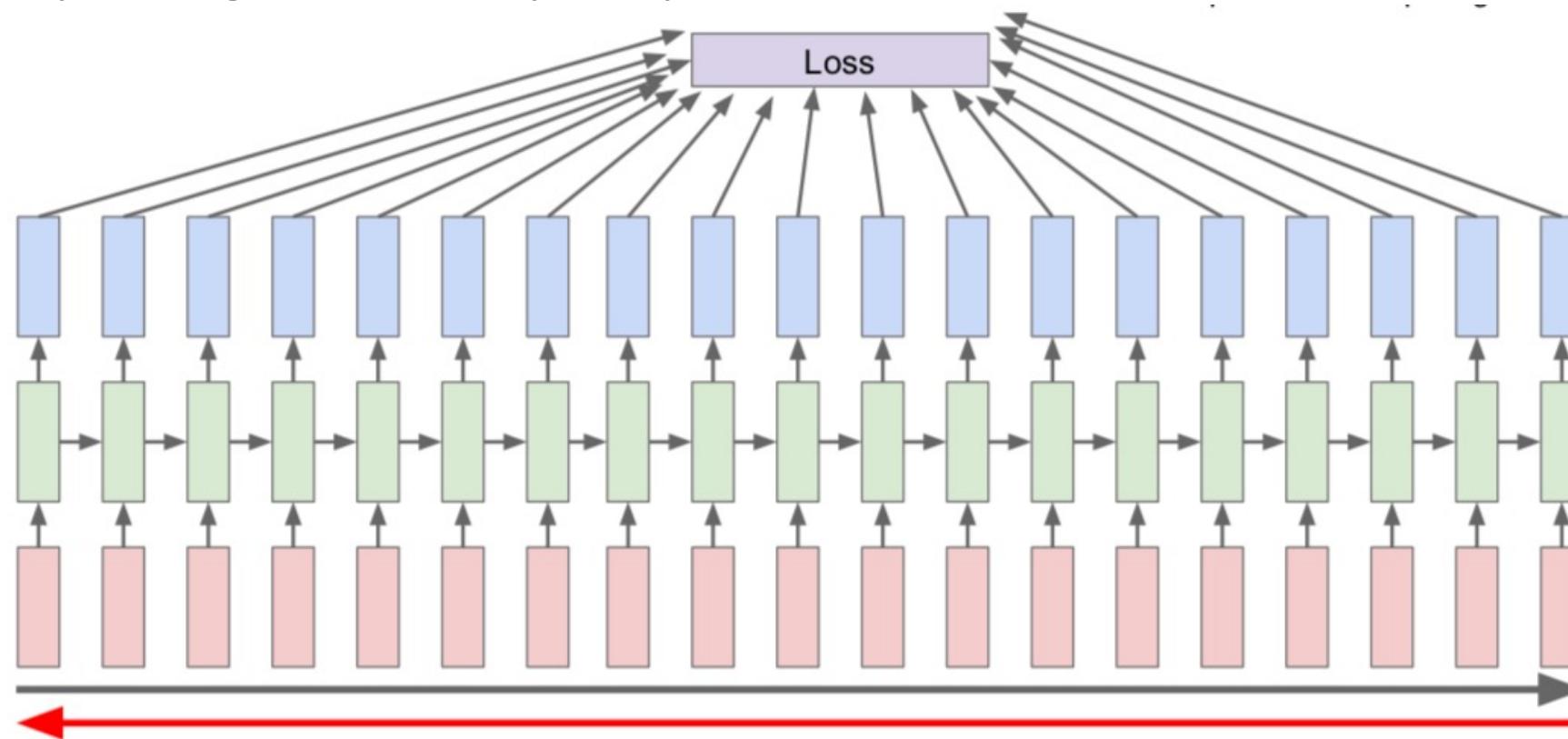
BPTT: Backpropagation

3. Get the gradients for all weights, and update the matrices using gradient descent.



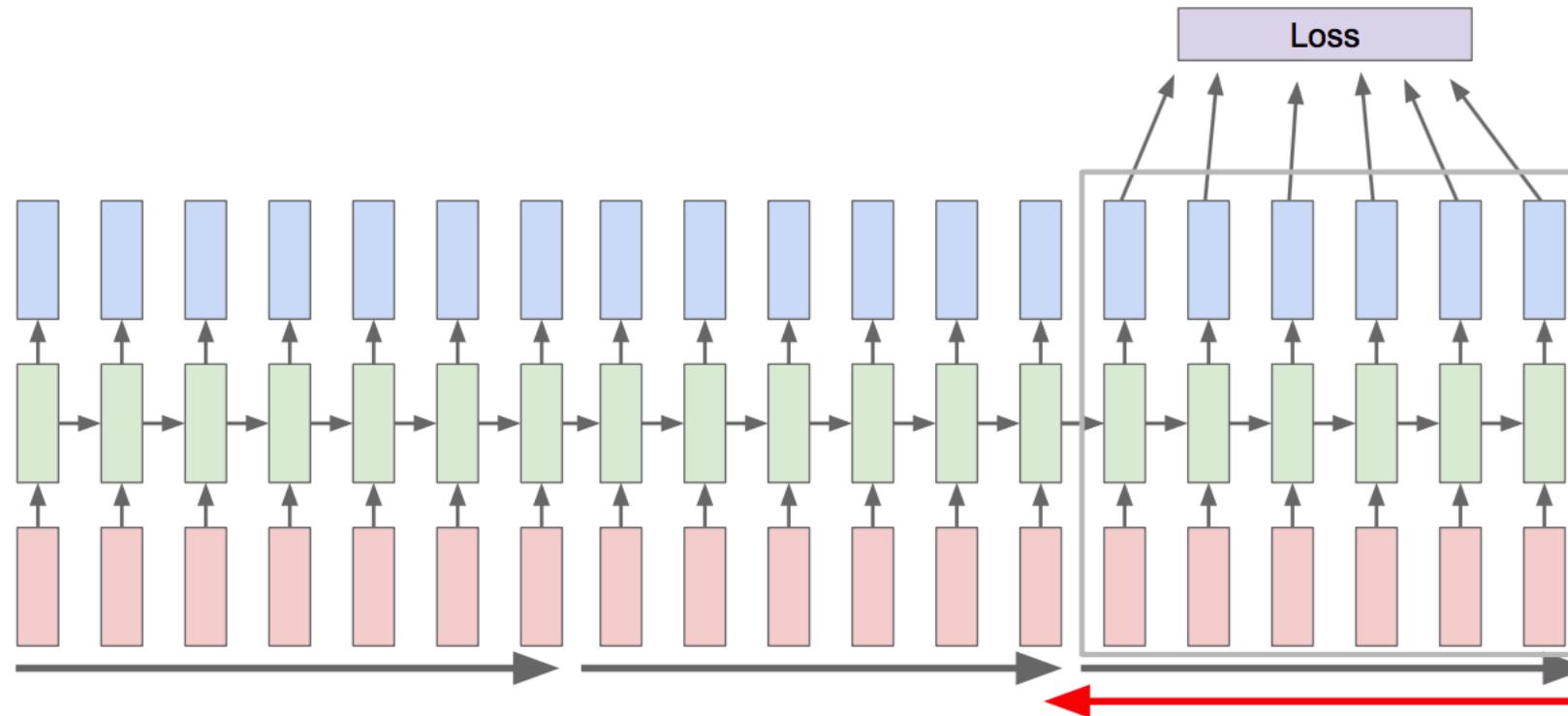
BPTT: Limitations

BPTT can be computationally very expensive as a lot of partial derivatives have to be computed, depending on the complexity of the network.



Truncated Backpropagation Through Time (Trunc-BPTT)

With the Truncated Backpropagation Through Time (**Trunc-BPTT**), instead of passing the whole sequence, we perform the forward and backward pass on a subset.





Deep Learning for Time Series – Recurrent models

Recap



In this lecture

- Deep learning
 - Perceptron
 - Layers
 - MLP
 - Gradient Descent
 - Backpropagation
- Recurrent neural network
 - Model
 - Architectures
- Backpropagation through time
 - BPTT
 - Trunc-BPTT

RNNs pros and cons

Pros:

- Regardless of the sequence length, the learned model always has the same input size
 - They perform better on dataset with sequences of variable-length.
- It is possible to use same transition for all time steps.

Cons:

- Vanishing/Exploding gradient



Machine Learning for Time Series

(MLTS or MLTS-Deluxe Lectures)

Dr. Dario Zanca

Machine Learning and Data Analytics (MaD) Lab
Friedrich-Alexander-Universität Erlangen-Nürnberg
24.01.2023

- Time series fundamentals and definitions (2 lectures)
- Bayesian Inference (1 lecture)
- Gaussian processes (2 lectures)
- State space models (2 lectures)
- Autoregressive models (1 lecture)
- Data mining on time series (1 lecture)
- Deep learning on time series (4 lectures) ←
- Domain adaptation (1 lecture)

In this lecture...

- 1. Long-term dependencies**
- 2. Long-short term memory networks (LSTMs)**
- 3. Other gated architectures**



Deep Learning for Time Series – Gated Models

Long-term dependencies



Representation ability of RNNs

RNNs are well suited for tasks involving sequences because they have an internal state that can represent temporal context information.

- The cycles in the graph of an RNN allow to keep information about input where the amount of time (i.e., time steps) is not fixed *a priori*.
- In contrast, feedforward neural networks (FF-NNs) have a “finite impulse response” and can not store information for an indefinite time.
- Parameters of a RNNs can be learned with BPTT, based on a cost function.

Long-term dependencies

Definition. A task is displaying **long-term dependencies** if the prediction of the desired output at time t depends on input presented at an earlier time $\tau \ll t$.

- In presence of long-term dependencies, RNNs can outperform a static FF-NN, although they appear **more difficult to train** optimally.
- Generally, their parameters tend to settle in **sub-optimal solutions that take into account short-term dependences**, but not long-term dependencies.

Learning long-term dependencies

For a parametric dynamical system that can learn to store relevant state information, we require the following:

1. The system is able to store information for an arbitrary duration (latching information)
2. The system is resistant to noise which is irrelevant to predicting a correct output (robustness).
3. The system's parameters are trainable (learnability).

Example: minimal task with long-term dependencies

The following minimal task can be regarded as a test which must be passed in order to satisfy (1), (2), and (3).

Minimal task (for testing a system learning long-term dependencies).

A parametric system is trained to classify two different sets of sequences of an arbitrary length T . For each sequence $s = (s_1, \dots, s_T)$, the corresponding class only depends on the initial L values, i.e.,

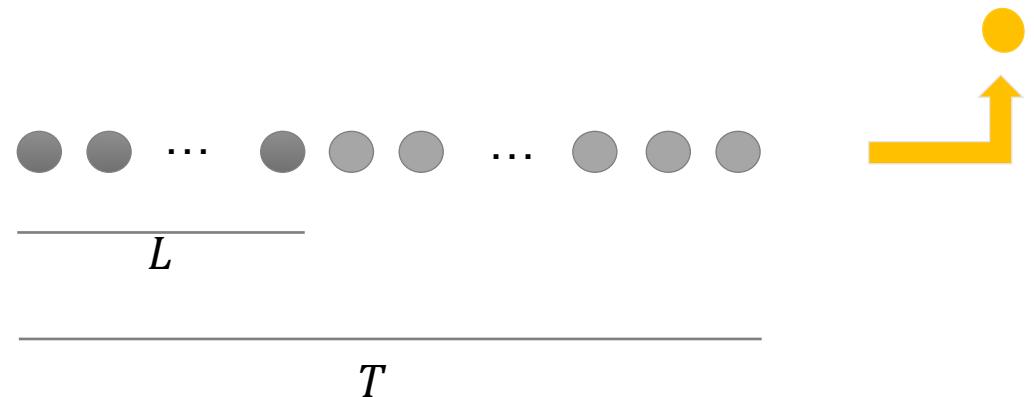
$$C(s_1, \dots, s_L, \dots, s_T) \equiv C(s_1, \dots, s_L)$$

where we suppose L is fixed and $L \ll T$. The system should provide a prediction at the end of each sequence.

Example: minimal task with long-term dependencies

The problem can be solved only if the system is capable of storing the information about the **initial input values** for an arbitrary duration.

The values (s_{L+1}, \dots, s_T) are irrelevant and can be regarded as *noise*. They can have the effect of erasing the internal information about the initial values of the input.



Example: minimal task with long-term dependencies

The test system has to process one input h_t and one state x_t for each discrete time step.

- The initial inputs h_t with $t \leq L$ contain the relevant information.
- We assume that h_t with $t > L$ is Gaussian noise.
- The connection weights of the test system are trainable.
- Optimization is based on the cost function

$$\mathcal{L} = \frac{1}{2} \sum_p (x_T^p - d^p)^2$$

where $d^p \in \{-1, 1\}$ represents the corresponding class of the input sequence.

Example: minimal task with long-term dependencies

A simple recurrent network candidate solution for the minimal tasks is made of a single recurrent neuron.

$$x_t^k = f(a_t^k) = \tanh(a_t^k)$$

$$a_t^k = w f(a_{t-1}^k) + h_t^k$$

$$a_0^0 = a_0^1 = 0$$

where k represents the corresponding sequence class.

It can be shown that, under mild conditions, the dynamic of this neuron system has two attractors and can robustly latch one bit of information represented by the sign of its activation.

Learning long-term dependencies is difficult

In order to latch a bit of information, the system must be able to restrict its activation to a subset S of its domain.

→ In this way it is possible to interpret the activation in two ways, as $a_t \in S$ or $a_t \notin S$.

To make sure the system remains in such a region, the system must be chosen in a way that the region is a basin of attraction of an hyperbolic attractor.

Two conditions can arise when using hyperbolic attractors to latch bits of infomations:

- Either the system is very sensitive to noise,
- Or the derivatives of the cost function at time t with respect to the system activation a_0 converge exponentially to 0.

Vanishing/Exploding gradient problem

Exploding gradient. When the gradients of the loss function with respect to the network parameters are very large.

- Learning is unstable
- A possible solution is **gradient clipping**, i.e., cutting off gradients which are greater than a threshold during backpropagation.

Vanishing gradient. When the gradients of the loss function with respect to the network parameters are very small.

- Learning is slow
- A possible solution is to introduce „shortcuts“ in the architecture, e.g., **gates**.

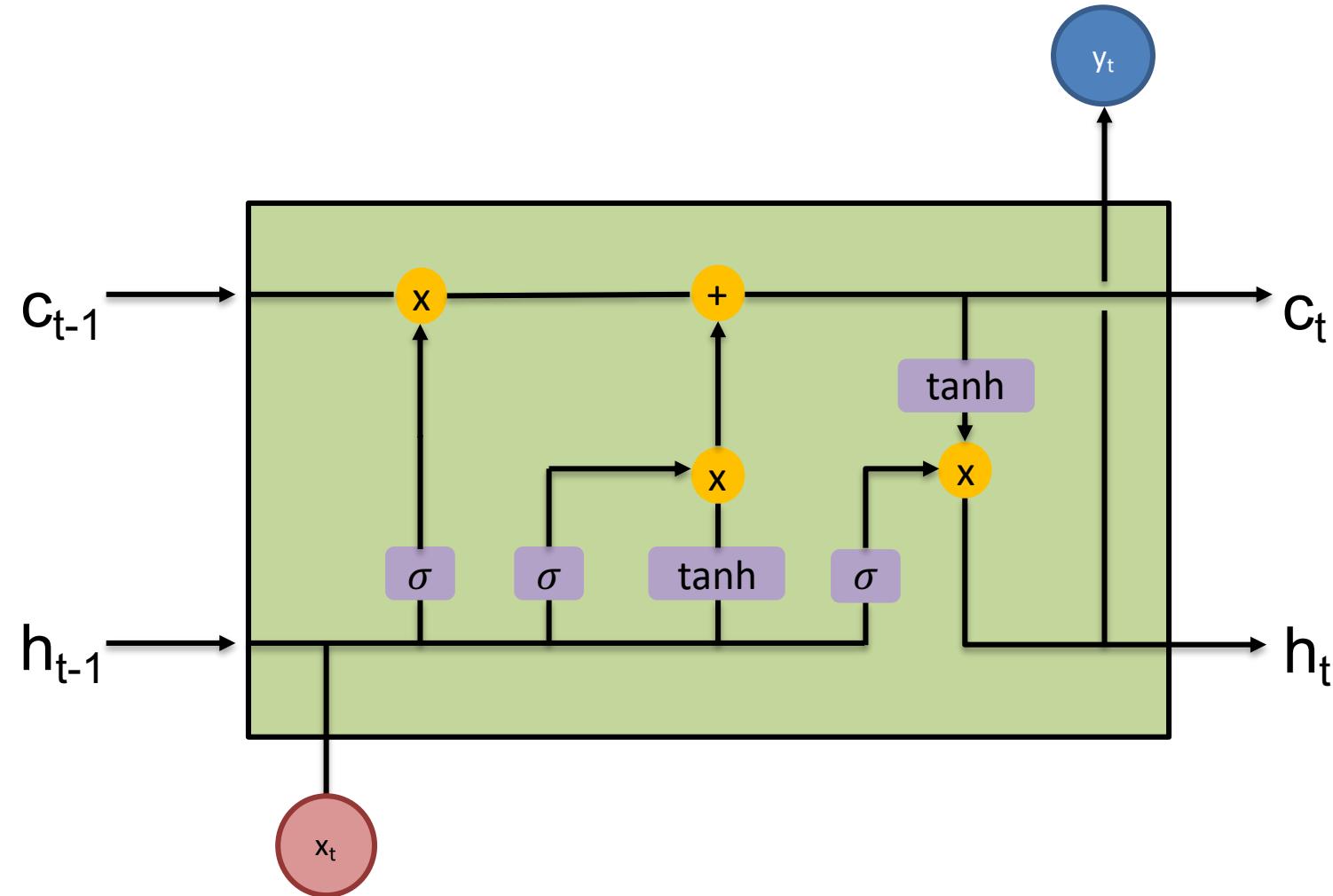


Deep Learning for Time Series – Gated Models

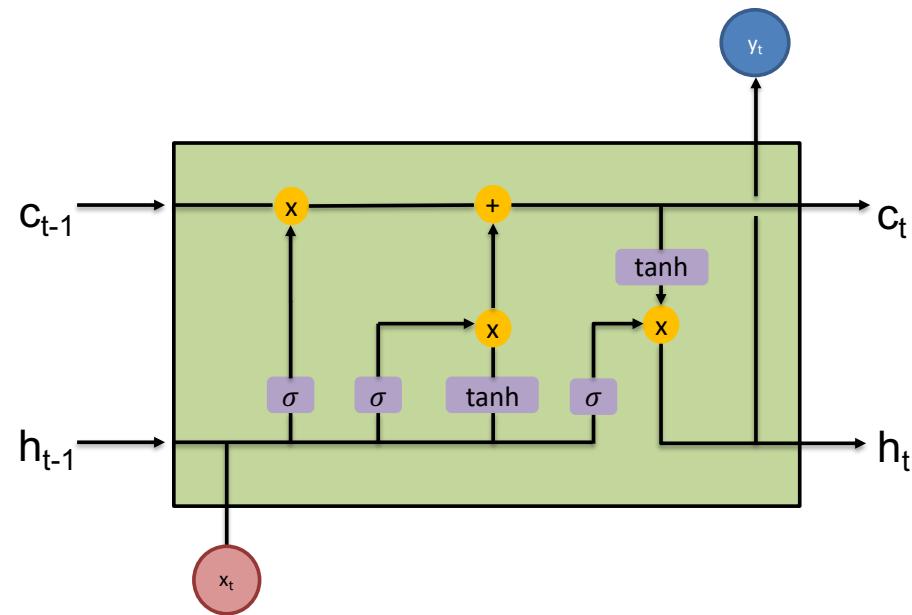
Long-short term memory networks (LSTMs)



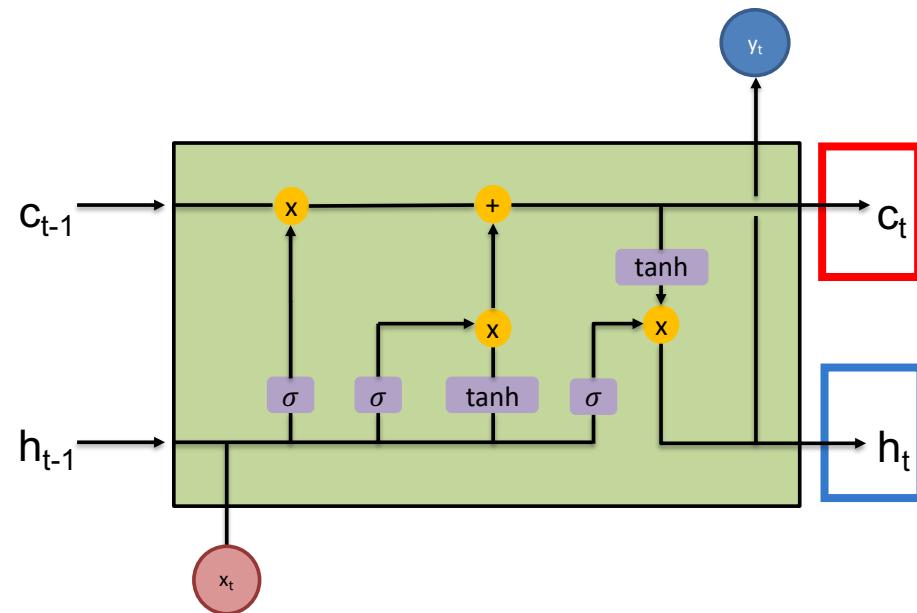
Long-short term memory networks (LSTMs)



Long-short term memory networks (LSTMs)



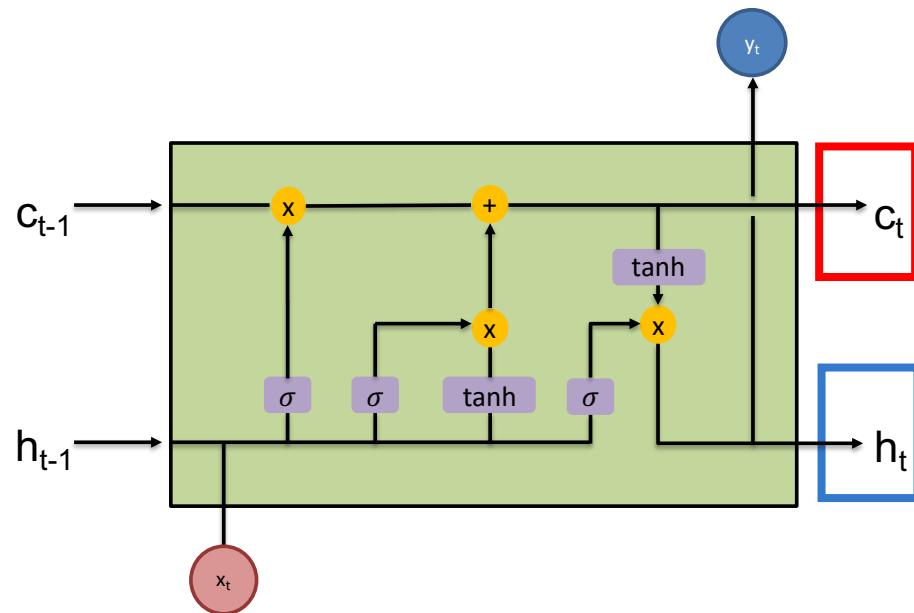
Long-short term memory networks (LSTMs)



Observations:

- Compared to RNNs, additionally to the **hidden state** h_t , we have another state, c_t , said the **cell state**.

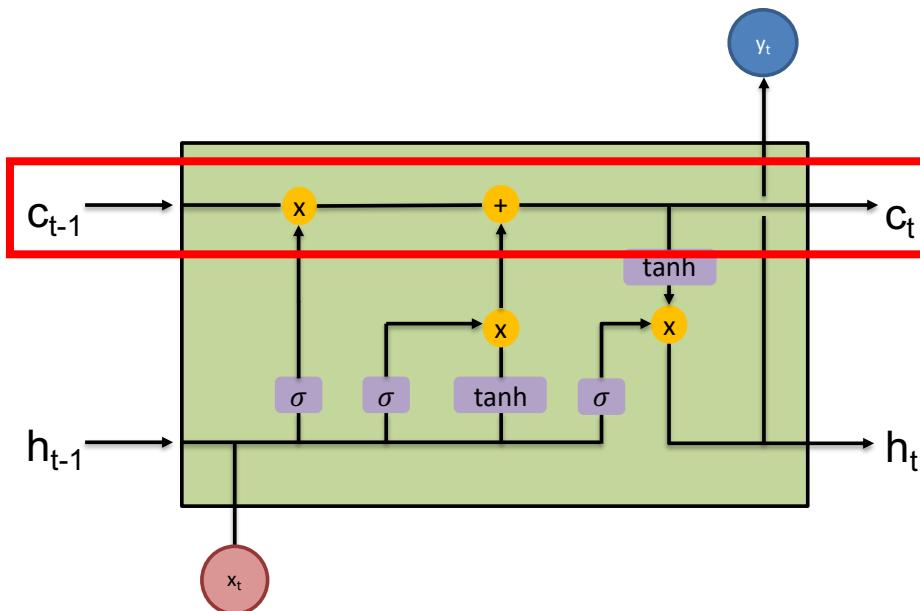
Long-short term memory networks (LSTMs)



Observations:

- Compared to RNNs, additionally to the **hidden state** h_t , we have another state, c_t , said the **cell state**.
- The information flow is regulated by **three gates**: the forget gate, the input gate and the output gate.

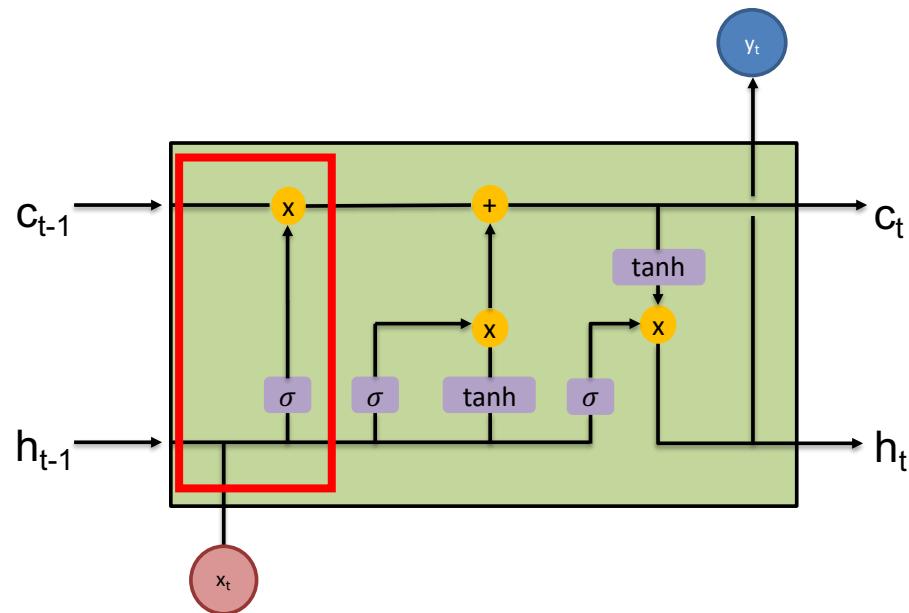
Long-short term memory networks (LSTMs)



The cell state has:

- Only minor interactions
- Simple information flow
- Other gates regulates whether it is preserved/not-preserved or updated.

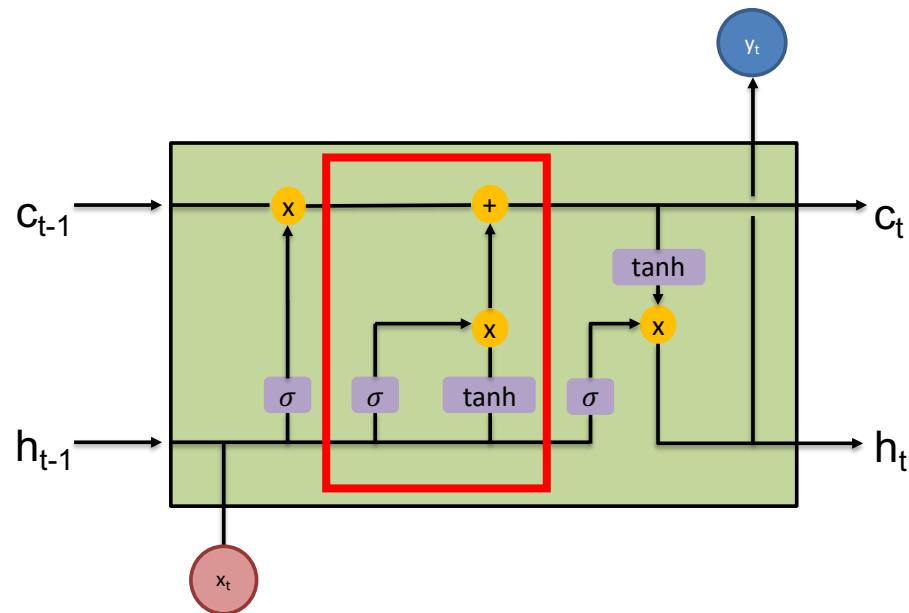
Long-short term memory networks (LSTMs)



The **forget gate** decides how much information to retain from the previous cell state.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

Long-short term memory networks (LSTMs)



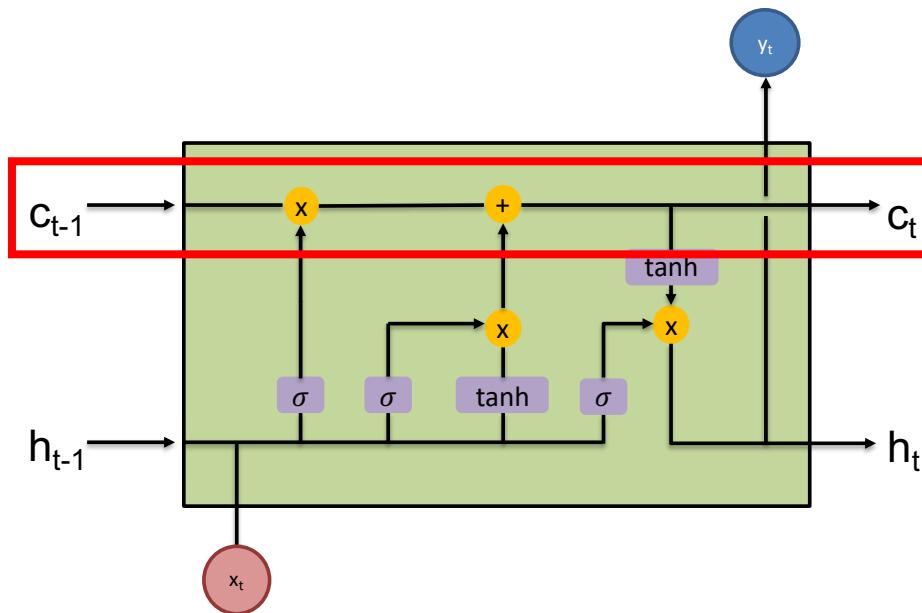
The **input gate** decides the information to be added to the cell state, based on the current input.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$g_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$

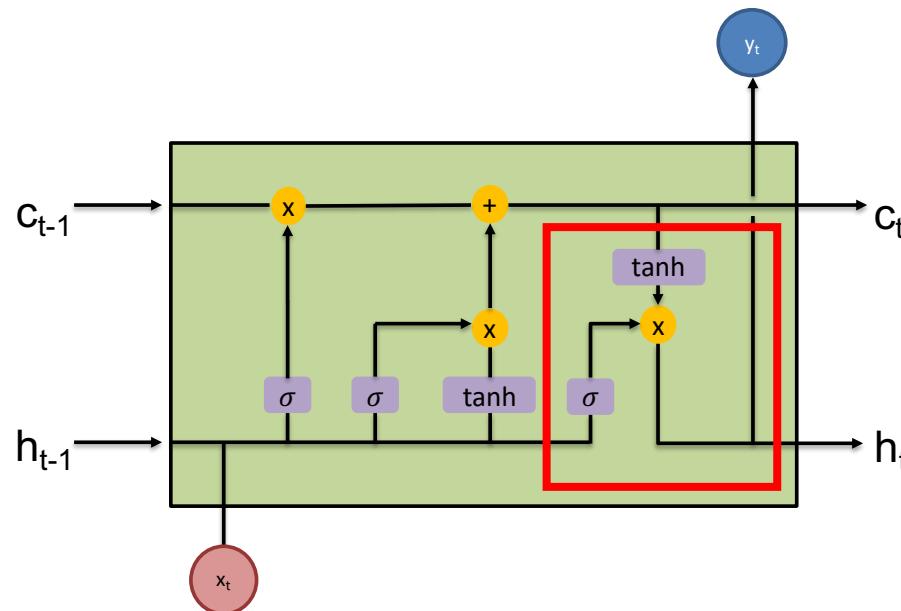
Long-short term memory networks (LSTMs)

The values can be combined to update the cell state



$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$

Long-short term memory networks (LSTMs)

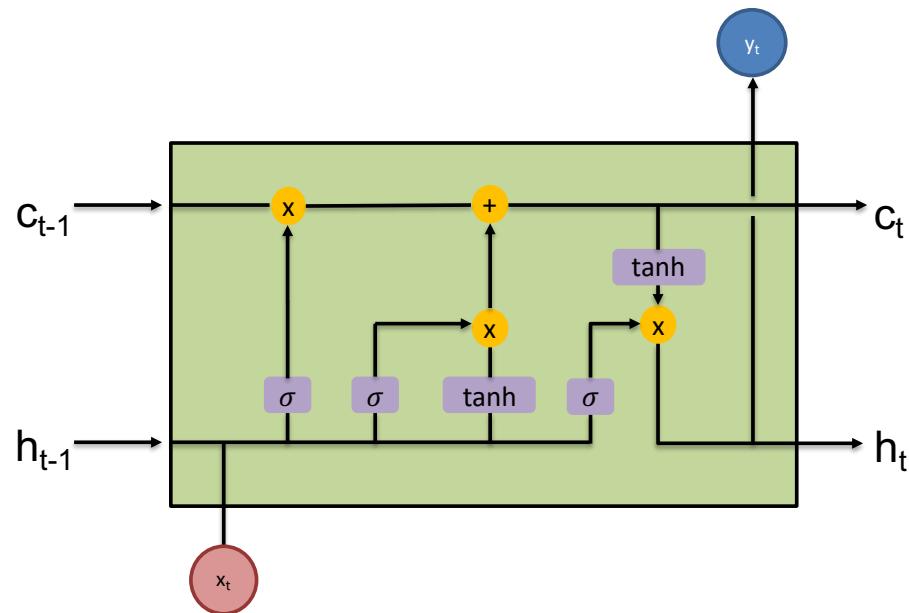


The **output gate** generates the output of the current LSTM cell, based on the current input, the previous output, and the updated cell state.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(c_t)$$

Long-short term memory networks (LSTMs)



To summarize, the LSTM cell is described by the following equations:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

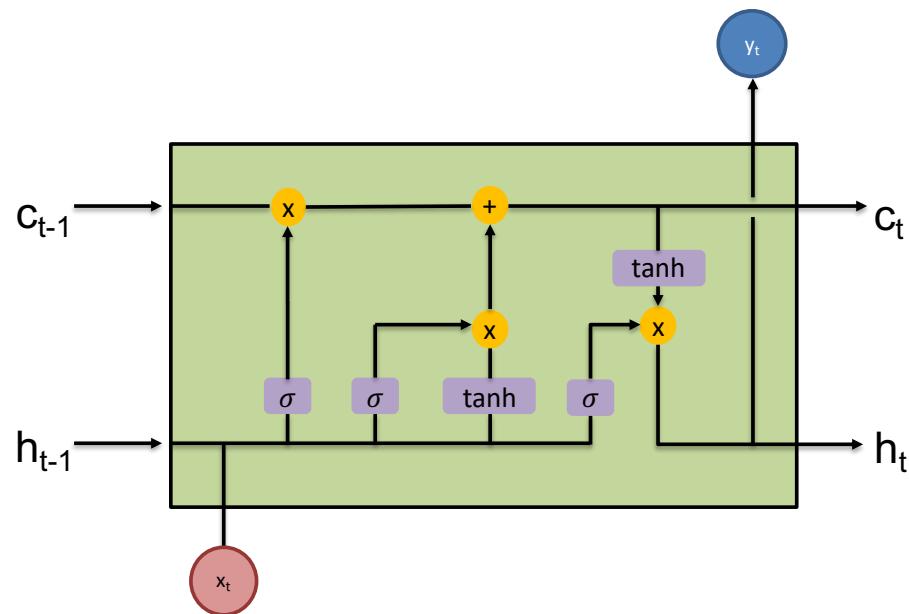
$$g_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(c_t)$$

Long-short term memory networks (LSTMs)



From the original publication [1]:

“Each memory cell’s internal architecture guarantees constant error flow within its constant error carrousel CEC... This represents the basis for bridging very long time lags. Two gate units learn to open and close access to error flow within each memory cell’s CEC. The multiplicative input gate affords protection of the CEC from perturbation by irrelevant inputs. Likewise, the multiplicative output gate protects other units from perturbation by currently irrelevant memory contents.”



Deep Learning for Time Series – Gated Models

Other gated architectures



Motivation

We can define different gated recurrent neural network architectures, e.g.,

- Fit a specific problem
- Reduce computations
- Adapt to different data characteristics

Alternatives to LSTMs:

- Gated recurrent unit (GRU)
- Phased-LSTM

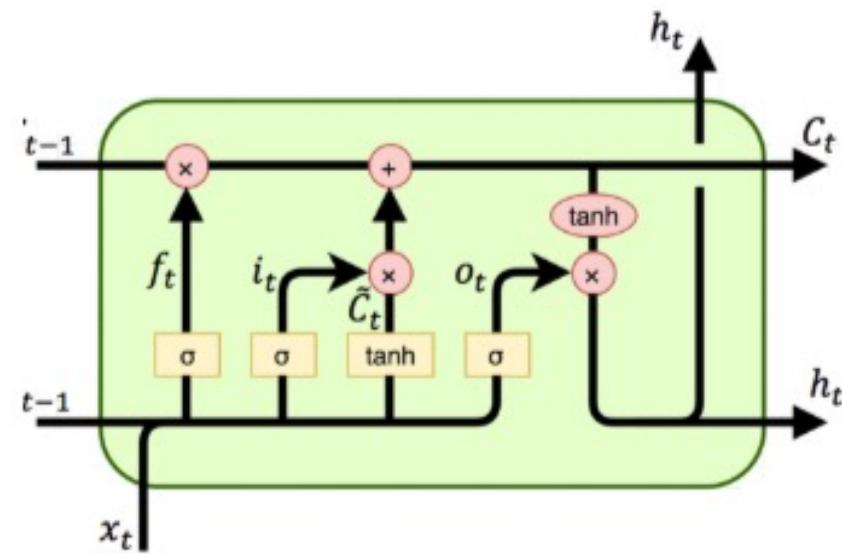
Gated recurrent units (GRUs)

Gated recurrent units (GRUs) are newer (published in 2014) architecture and can be seen a simplification of LSTM (published in 1997).

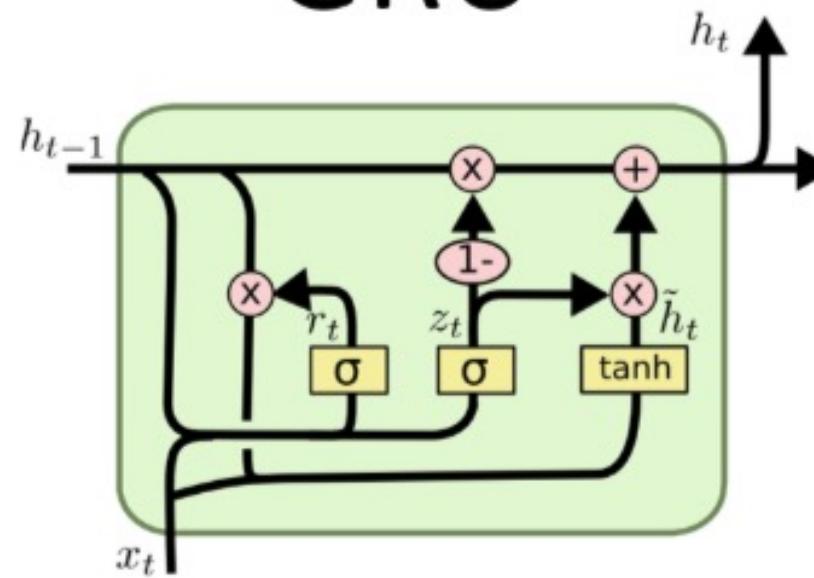
- GRU is composed of only two gates: the reset gate and the update gate.
- It propagates a single state
- It's, then, represented by fewer equations
 - Less parameters to learn
 - Faster to train

GRU vs LSTM

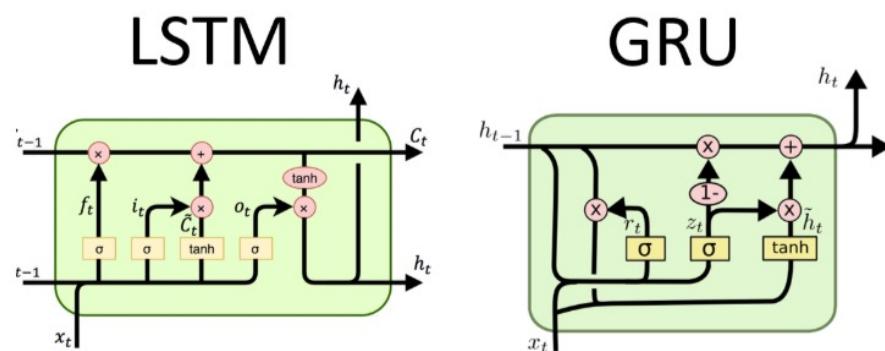
LSTM



GRU



GRU vs LSTM



When do I prefer GRU over LSTM?

- It depends on the data
- It needs empirical evaluation

In this empirical study [2], GRU outperformed the LSTM on all tasks with the exception of language modelling.

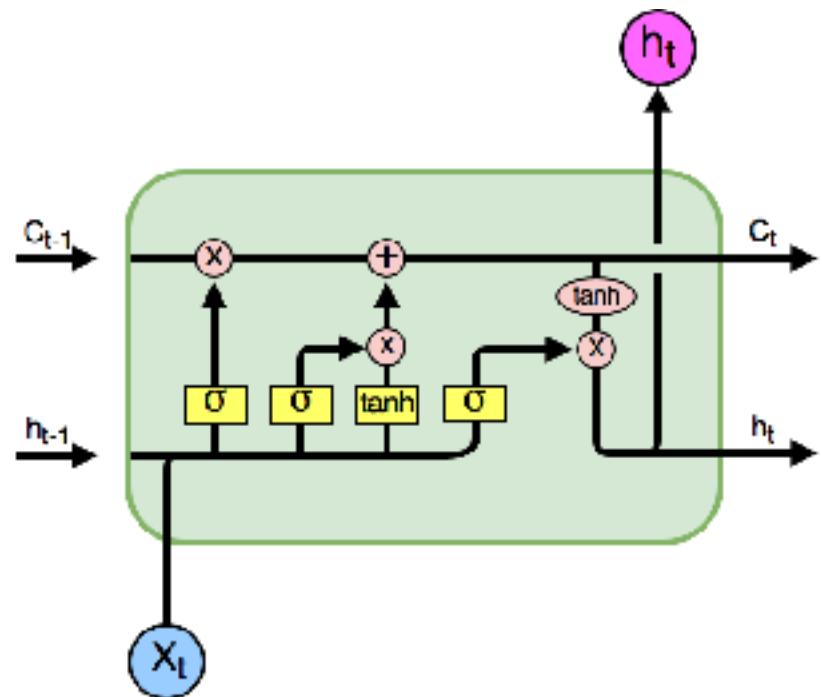
Phased-LSTM

Phased-LSTM [3] is designed to deal with input sampled at asynchronous times.

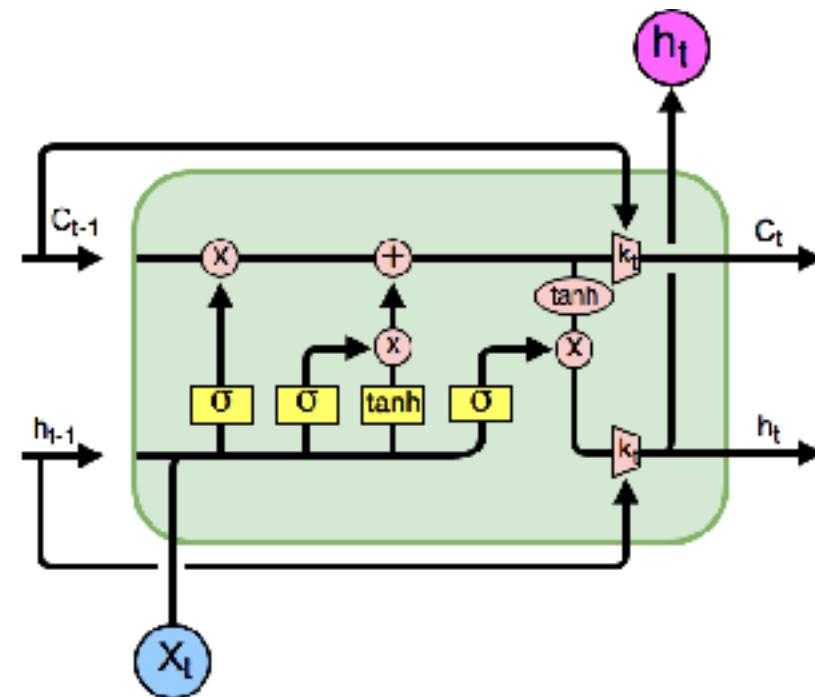
- Extends the LSTM unit by adding a new time gate.
- This gate is controlled by a parametrized oscillation with a frequency range that produces updates of the memory cell only during a small percentage of the cycle.
- The model naturally integrates inputs from sensors of arbitrary sampling rates.

[3] Neil, D., Pfeiffer, M., & Liu, S. C. (2016). Phased lstm: Accelerating recurrent network training for long or event-based sequences. *arXiv preprint arXiv:1610.09513*.

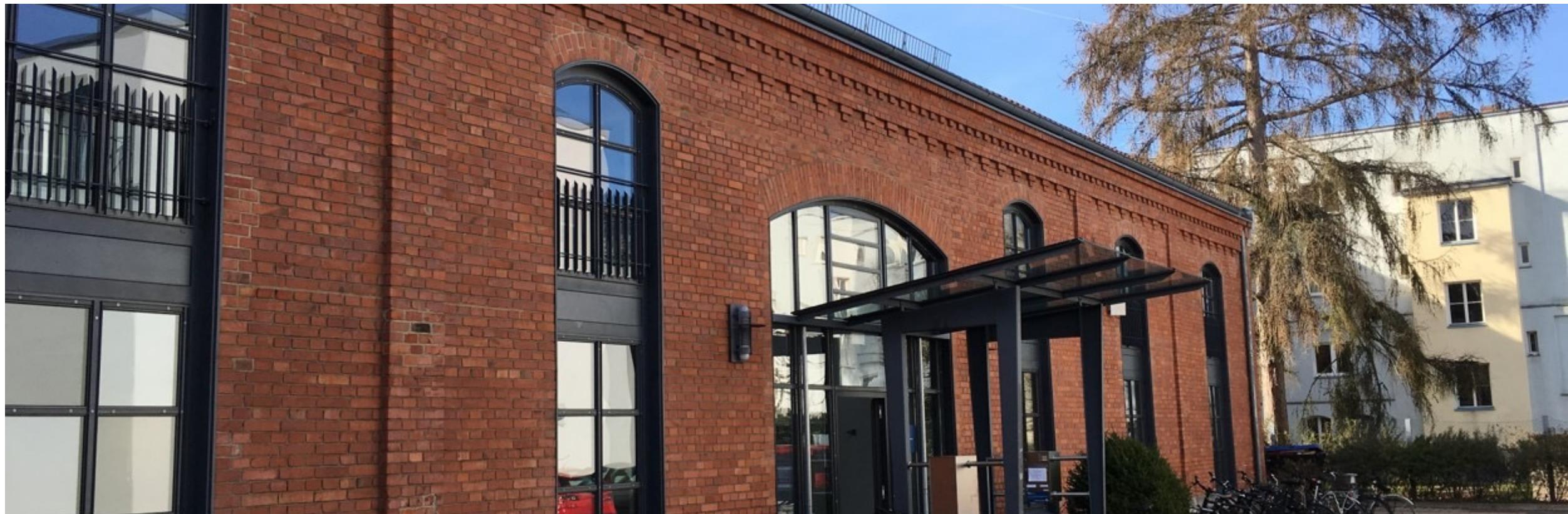
Phased-LSTM vs LSTM



(a) Standard LSTM



(b) Phased LSTM



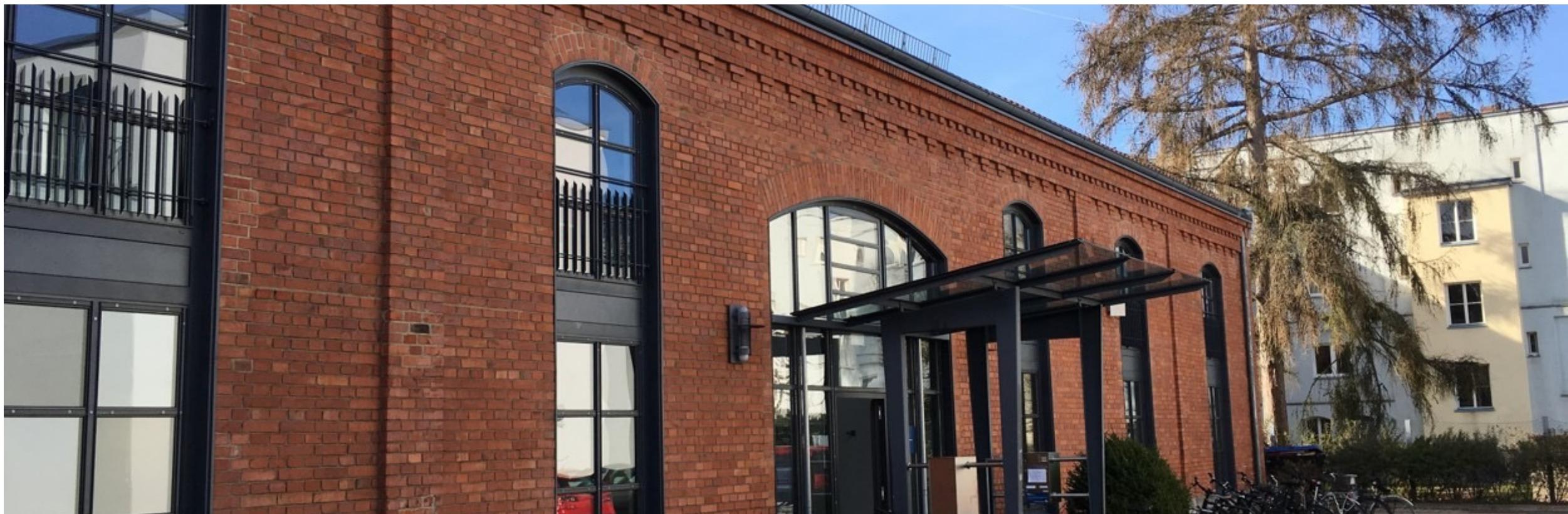
Deep Learning for Time Series – Gated Models

Recap



In this lecture

- Long-term dependencies
 - Exploding/Vanishing gradient
- Long-short term memory networks (LSTMs)
- Other gated architectures
 - GRU
 - Phased-LSTM



Machine Learning for Time Series

(MLTS or MLTS-Deluxe Lectures)

Dr. Dario Zanca

Machine Learning and Data Analytics (MaD) Lab
Friedrich-Alexander-Universität Erlangen-Nürnberg
31.01.2023

- Time series fundamentals and definitions (2 lectures)
- Bayesian Inference (1 lecture)
- Gaussian processes (2 lectures)
- State space models (2 lectures)
- Autoregressive models (1 lecture)
- Data mining on time series (1 lecture)
- Deep learning on time series (4 lectures) ←
- Domain adaptation (1 lecture)

In this lecture...

- 1. Convolutional neural networks**

- 2. Convolution-based architectures**



Deep Learning for Time Series – Convolutional Models

Convolutional Neural Networks (CNNs)



Motivation

RNN / LSTM limitations:

- Non-parallelism → Long training time
- Difficulties with long sequences
 - Large memory usage
 - Difficult to train (vanishing/exploding gradients)
 - Hard to learn long-term dependencies (mitigated by LSTMs)

Some of this problems are attenuated on CNNs:

- Parallel computations
- Easy to use on large input data

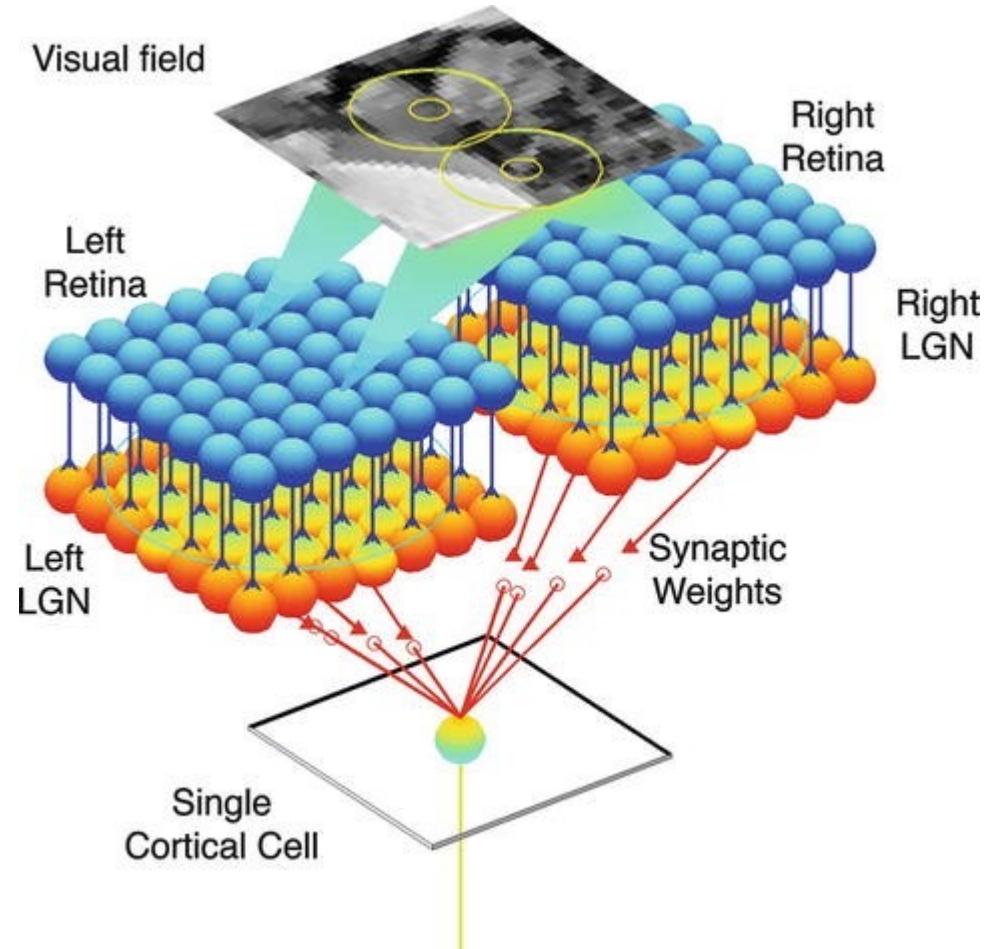
Human visual cortex

CNNs were inspired by the internal functioning of a specialized brain area: the visual cortex.

The visual cortex is in charge of processing visual information collected by the retinae.

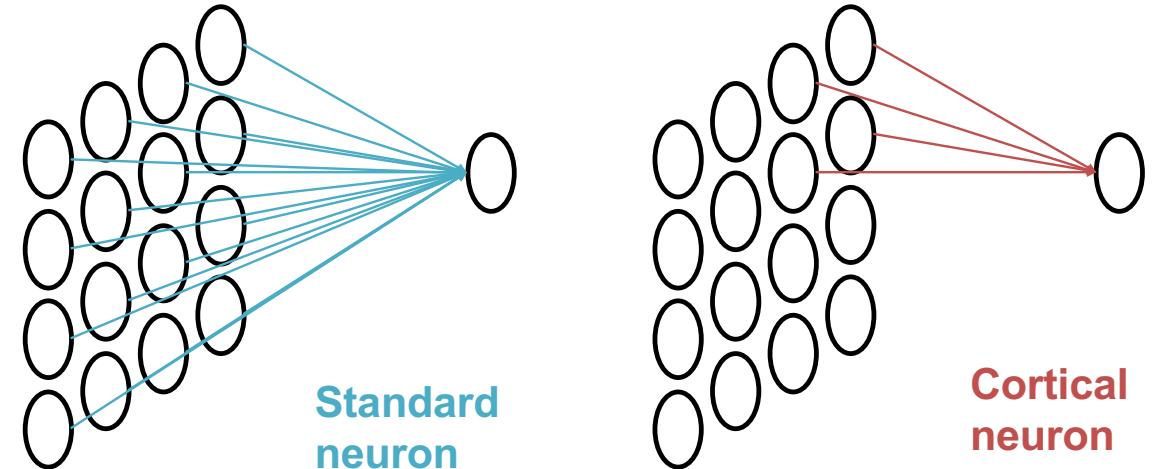
Cortical neurons only respond to a small portion of the stimuli (small receptive field).

Small receptive fields are stimulated by high spatial frequencies (fine details); large receptive fields are stimulated by low spatial frequencies (coarse details).



Modeling small receptive fields

- In standard multilayer perceptrons, neurons are connected to all units from the previous layer.
- Cortical neurons have small receptive fields: they have sparse and localized connection with units from the previous layer.
→ This is modeled by means of convolutional operations.



Convolutional operation

The (discrete) convolution is a mathematical operation on two functions (in our case, the input and a smaller filter) that produces a third function (the feature map).

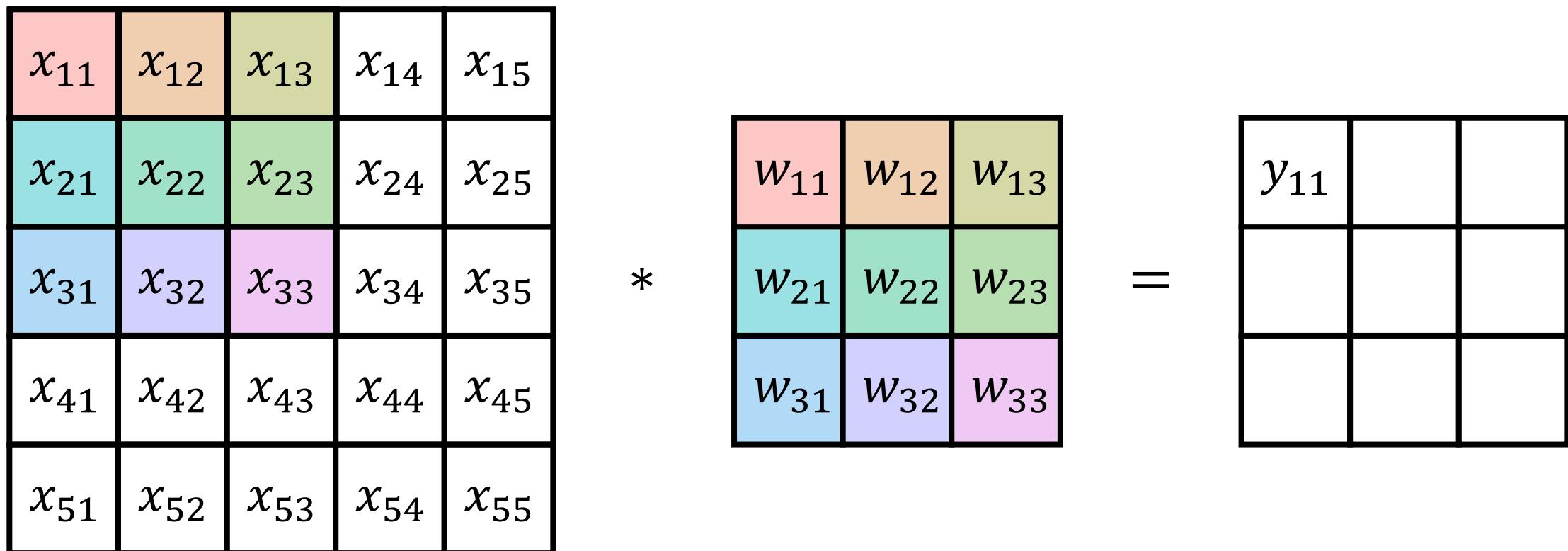
$$(input * filter)[n] = \sum_{m=-\infty}^{+\infty} input[m] \cdot filter[n - m]$$

- The sum is evaluated for all values of the shift.
- The term convolution is used both to indicate the result function and process of computing it.

Convolutional operation (2D)

The output computation now only depends on a subset of inputs:

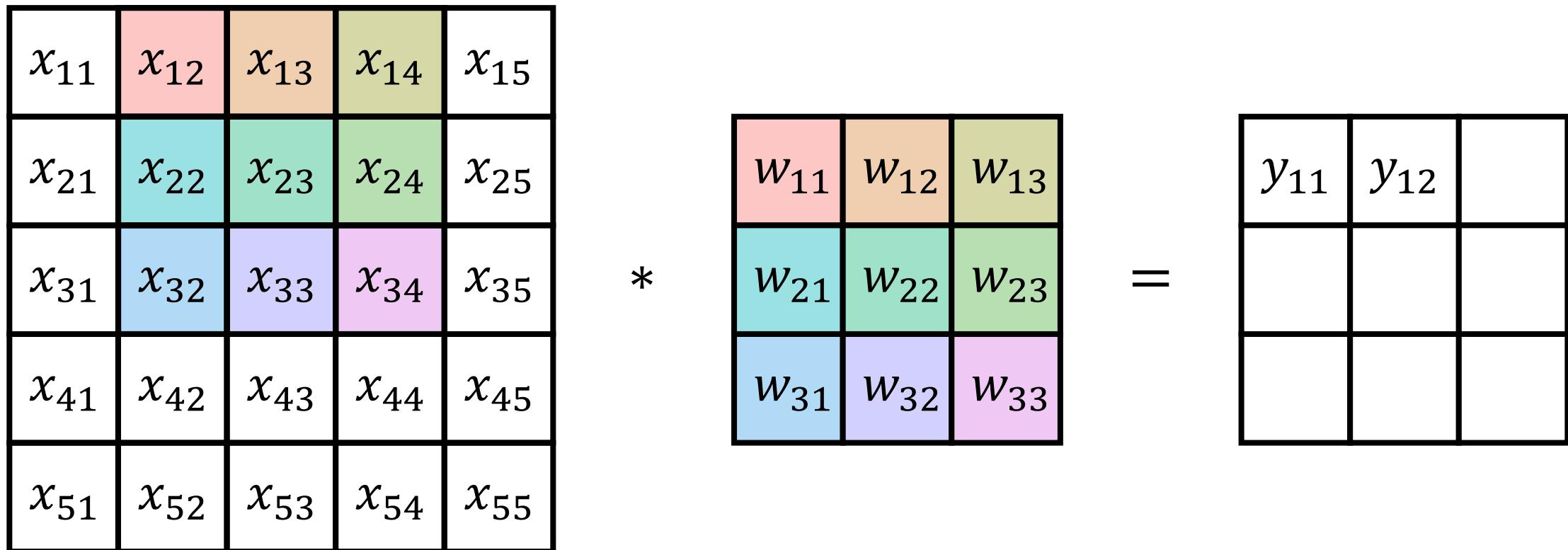
$$y_{11} = w_{11}x_{11} + w_{12}x_{12} + w_{13}x_{13} + w_{21}x_{21} + w_{22}x_{22} + w_{23}x_{23} + w_{31}x_{31} + w_{32}x_{32} + w_{33}x_{33}$$



Convolutional operation (2D)

The output computation now only depends on a subset of inputs:

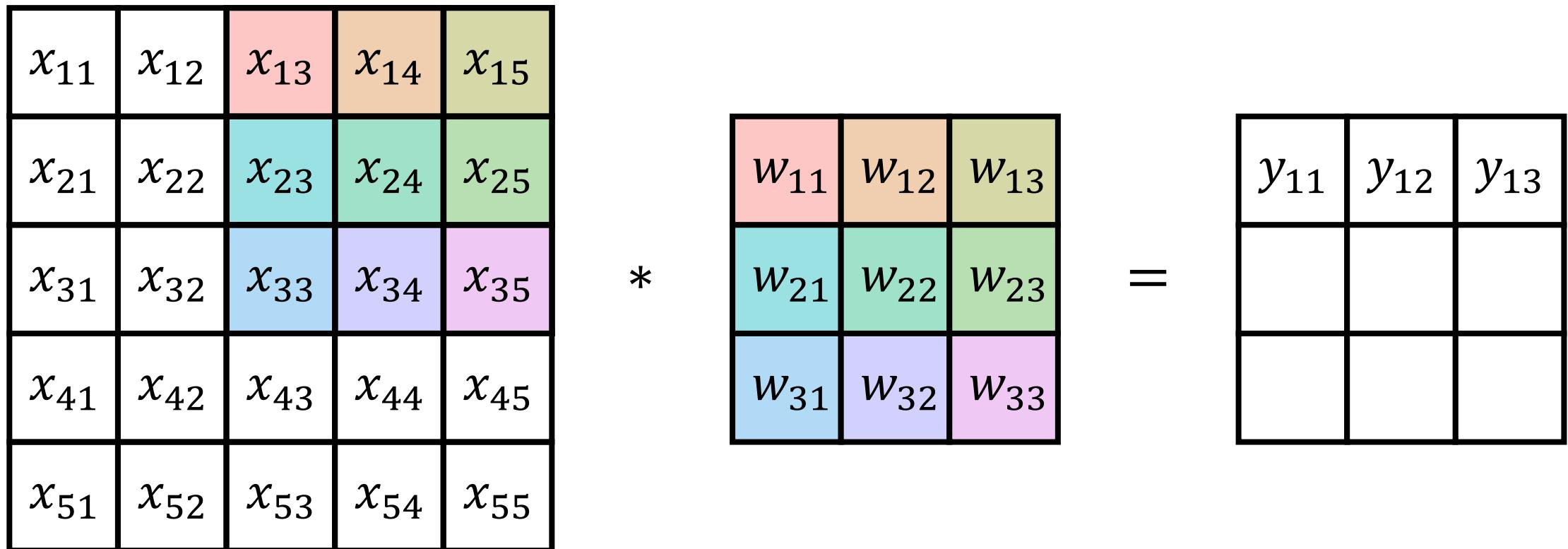
$$y_{12} = w_{11}x_{12} + w_{12}x_{13} + w_{13}x_{14} + w_{21}x_{22} + w_{22}x_{23} + w_{23}x_{24} + w_{31}x_{32} + w_{32}x_{33} + w_{33}x_{34}$$



Convolutional operation (2D)

The output computation now only depends on a subset of inputs:

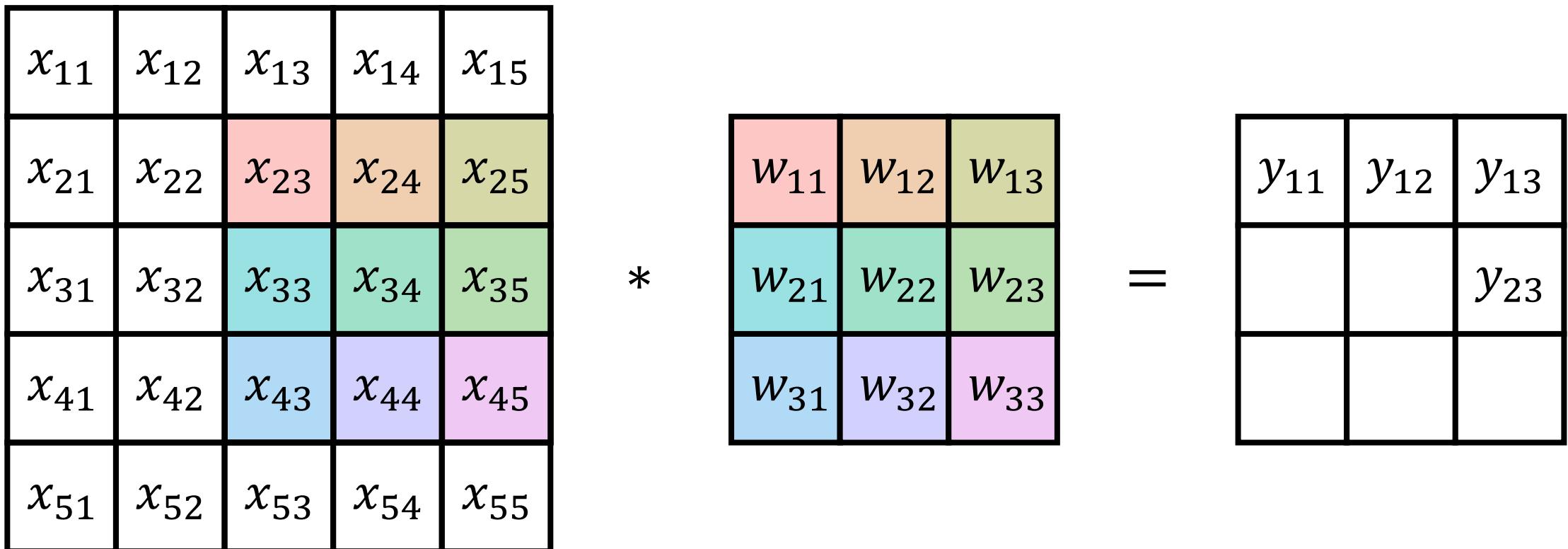
$$y_{13} = w_{11}x_{13} + w_{12}x_{14} + w_{13}x_{15} + w_{21}x_{23} + w_{22}x_{24} + w_{23}x_{25} + w_{31}x_{33} + w_{32}x_{34} + w_{33}x_{35}$$



Convolutional operation (2D)

The output computation now only depends on a subset of inputs:

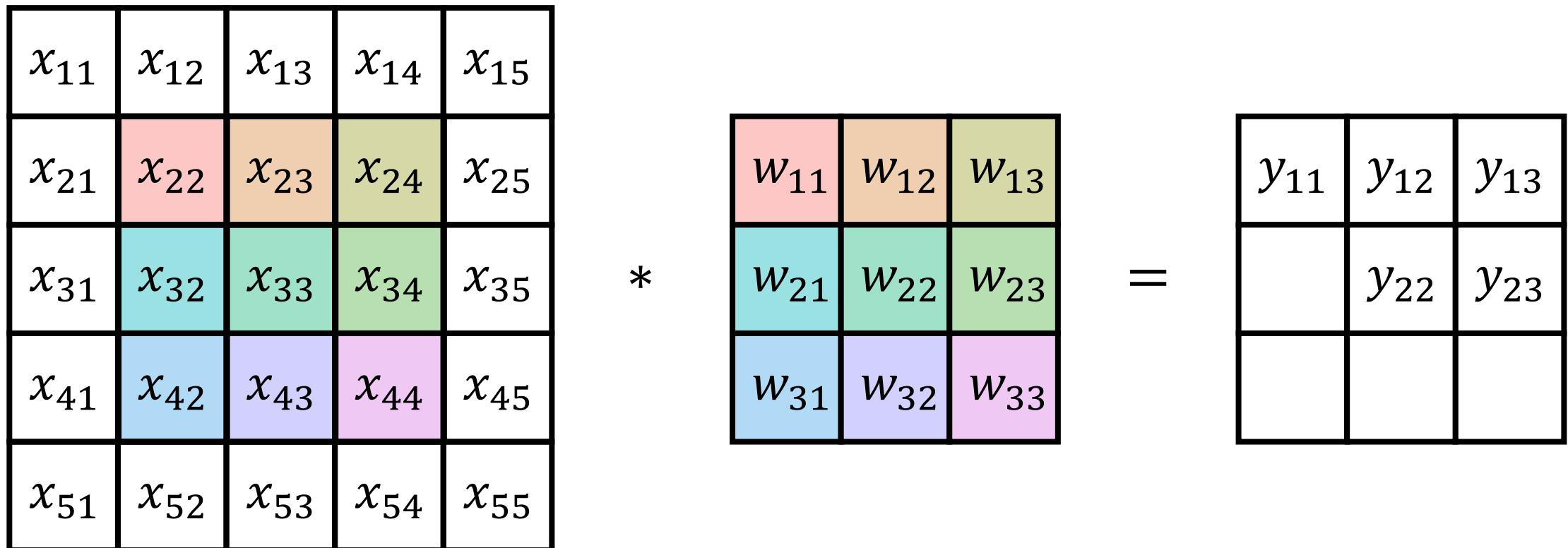
$$y_{23} = w_{11}x_{23} + w_{12}x_{24} + w_{13}x_{25} + w_{21}x_{33} + w_{22}x_{34} + w_{23}x_{35} + w_{31}x_{43} + w_{32}x_{44} + w_{33}x_{45}$$



Convolutional operation (2D)

The output computation now only depends on a subset of inputs:

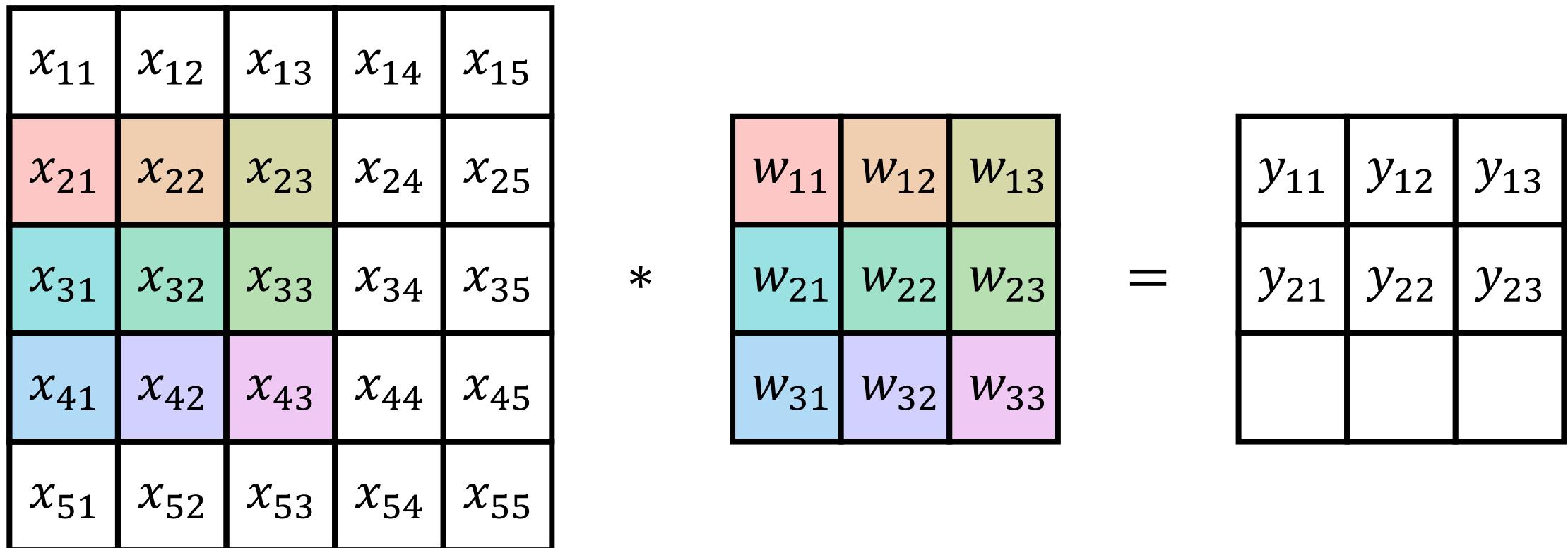
$$y_{22} = w_{11}x_{22} + w_{12}x_{23} + w_{13}x_{24} + w_{21}x_{32} + w_{22}x_{33} + w_{23}x_{34} + w_{31}x_{42} + w_{32}x_{43} + w_{33}x_{44}$$



Convolutional operation (2D)

The output computation now only depends on a subset of inputs:

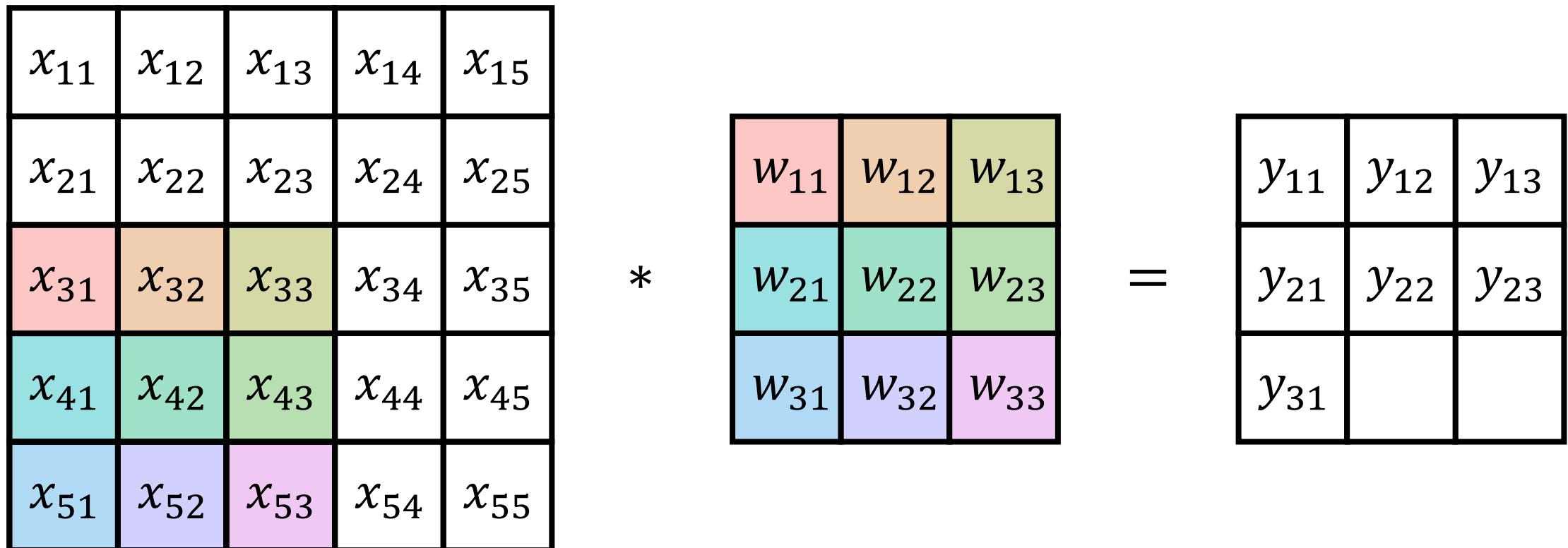
$$y_{21} = w_{11}x_{21} + w_{12}x_{22} + w_{13}x_{23} + w_{21}x_{31} + w_{22}x_{32} + w_{23}x_{33} + w_{31}x_{41} + w_{32}x_{42} + w_{33}x_{43}$$



Convolutional operation (2D)

The output computation now only depends on a subset of inputs:

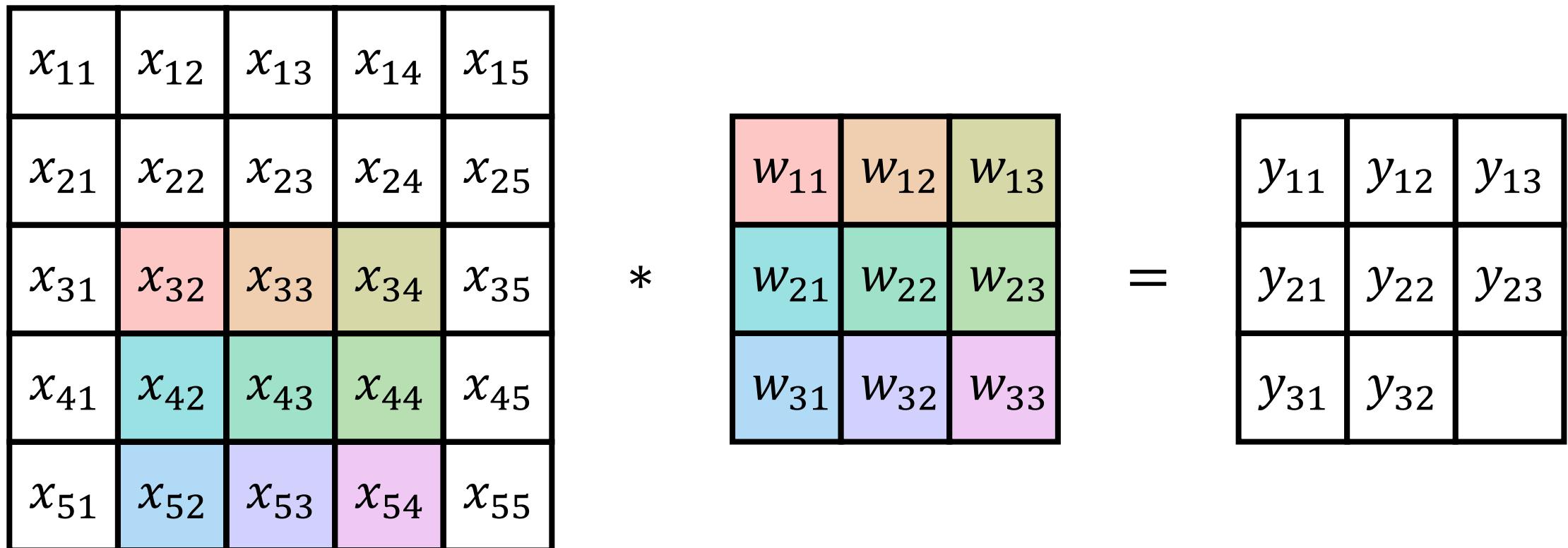
$$y_{31} = w_{11}x_{31} + w_{12}x_{32} + w_{13}x_{33} + w_{21}x_{41} + w_{22}x_{42} + w_{23}x_{43} + w_{31}x_{51} + w_{32}x_{52} + w_{33}x_{53}$$



Convolutional operation (2D)

The output computation now only depends on a subset of inputs:

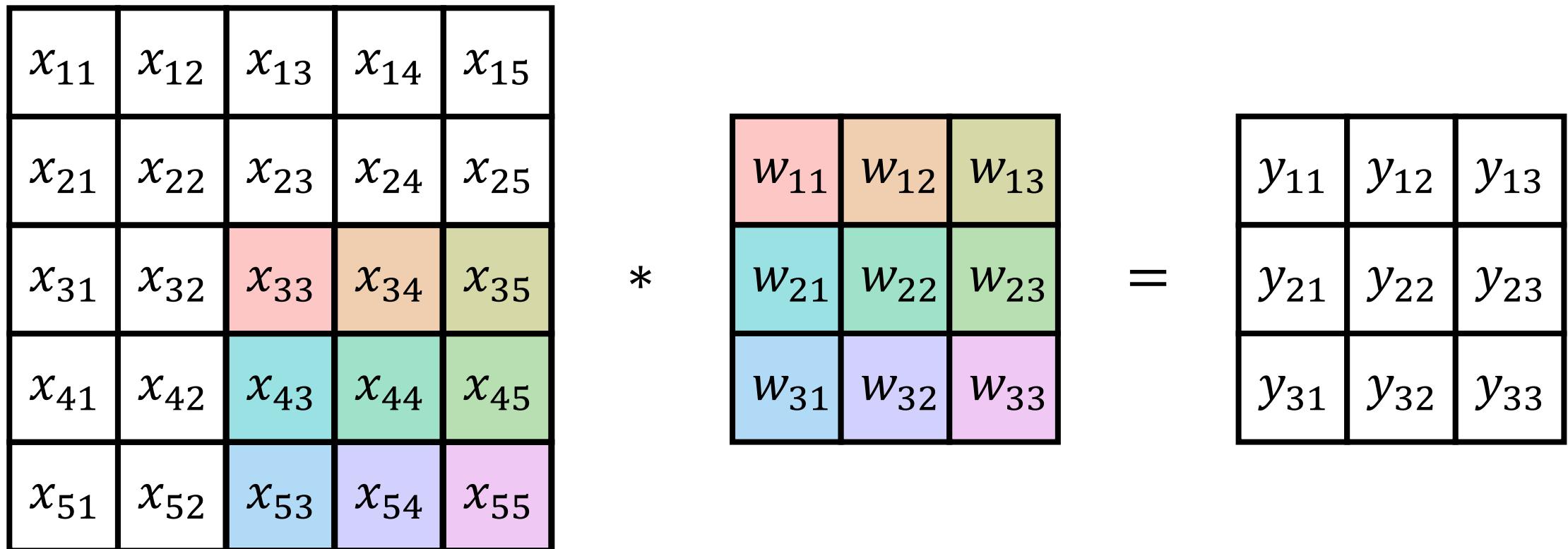
$$y_{32} = w_{11}x_{32} + w_{12}x_{33} + w_{13}x_{34} + w_{21}x_{42} + w_{22}x_{43} + w_{23}x_{44} + w_{31}x_{52} + w_{32}x_{53} + w_{33}x_{54}$$



Convolutional operation (2D)

The output computation now only depends on a subset of inputs:

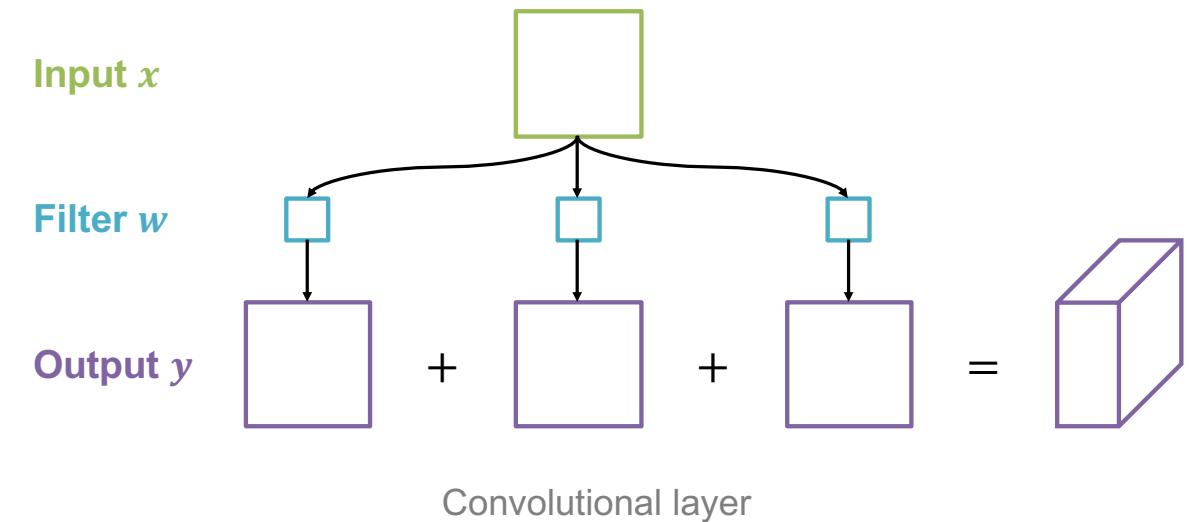
$$y_{33} = w_{11}x_{33} + w_{12}x_{34} + w_{13}x_{35} + w_{21}x_{43} + w_{22}x_{44} + w_{23}x_{45} + w_{31}x_{53} + w_{32}x_{54} + w_{33}x_{55}$$



Convolutional layer

In a convolutional layer, multiple filters can be used in parallel, which result in multiple convolutional operations in parallel on the same input.

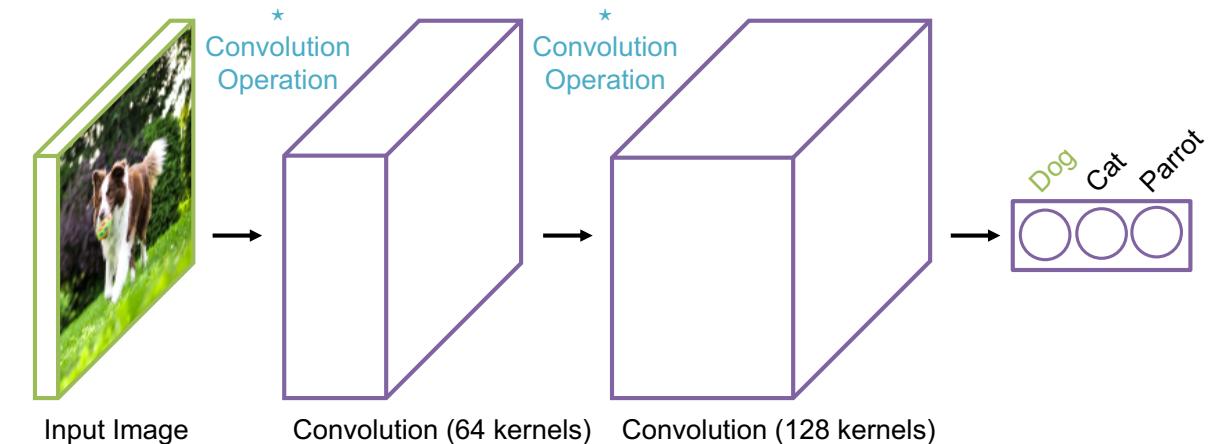
The different output are concatenated to create a multi-channel feature map.



Convolutional Neural Network (CNN)

To incrementally process relevant features, multiple layers of convolution can be stacked to create a deep convolutional neural network (CNN).

However, after many layer the number of features grow very quickly
→ Computationally expensive



Pooling operation

The pooling operation is introduced to reduce the number of computations in a CNN. From a theoretical perspective, higher representations do not require high spatial resolution.

$$y_{11} = \max(x_{11}, x_{12}, x_{21}, x_{22})$$

x_{11}	x_{12}	x_{13}	x_{14}
x_{21}	x_{22}	x_{23}	x_{24}
x_{31}	x_{32}	x_{33}	x_{34}
x_{41}	x_{42}	x_{43}	x_{44}

$\max x$
→

y_{11}	

Pooling operation

The pooling operation is introduced to reduce the number of computations in a CNN. From a theoretical perspective, higher representations do not require high spatial resolution.

$$y_{12} = \max(x_{13}, x_{14}, x_{23}, x_{24})$$

x_{11}	x_{12}	x_{13}	x_{14}
x_{21}	x_{22}	x_{23}	x_{24}
x_{31}	x_{32}	x_{33}	x_{34}
x_{41}	x_{42}	x_{43}	x_{44}

$\max x$
→

y_{11}	y_{12}

Pooling operation

The pooling operation is introduced to reduce the number of computations in a CNN. From a theoretical perspective, higher representations do not require high spatial resolution.

$$y_{21} = \max(x_{31}, x_{32}, x_{41}, x_{42})$$

x_{11}	x_{12}	x_{13}	x_{14}
x_{21}	x_{22}	x_{23}	x_{24}
x_{31}	x_{32}	x_{33}	x_{34}
x_{41}	x_{42}	x_{43}	x_{44}

$\max x$
→

y_{11}	y_{12}
y_{21}	

Pooling operation

The pooling operation is introduced to reduce the number of computations in a CNN. From a theoretical perspective, higher representations do not require high spatial resolution.

$$y_{22} = \max(x_{33}, x_{34}, x_{43}, x_{44})$$

x_{11}	x_{12}	x_{13}	x_{14}
x_{21}	x_{22}	x_{23}	x_{24}
x_{31}	x_{32}	x_{33}	x_{34}
x_{41}	x_{42}	x_{43}	x_{44}

$\max x$
→

y_{11}	y_{12}
y_{21}	y_{22}

Pooling operation

The pooling operation is introduced to reduce the number of computations in a CNN. From a theoretical perspective, higher representations do not require high spatial resolution.

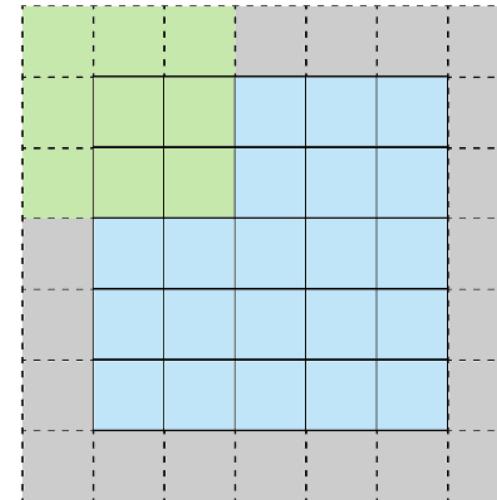
- Other operations can be used instead of the max, e.g., the mean, L^2 -norm, ...
- There is no general consensus on which of these operation is the best and this should be experimentally validated case by case.

Padding operation

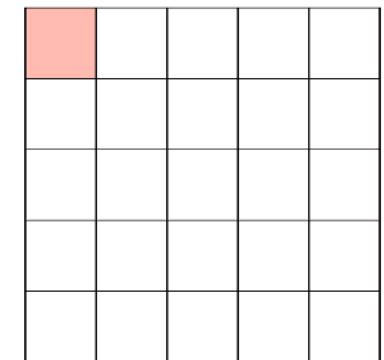
The convolution, as illustrated previously, leads to a reduction of the dimensionality.

→ Given an input of size n and a filter of size m , the resulting feature map have size $n - m + 1$.

For this reason, the padding operation is used to add symmetrically zeros to the input, such that the resulting feature map is of the same size as the input.



Stride 1 with Padding



Feature Map

1-D Convolutional Neural Networks

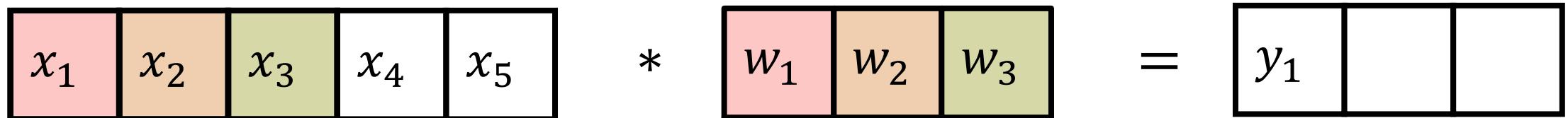
CNNs can be used to learn temporal dependencies on time series data.

- The convolution is applied along a single dimension, i.e., the temporal dimension.
- The resulting model is generally referred to as 1-D CNN.

Convolutional operation (1D)

The output computation now only depends on a subset of the time series:

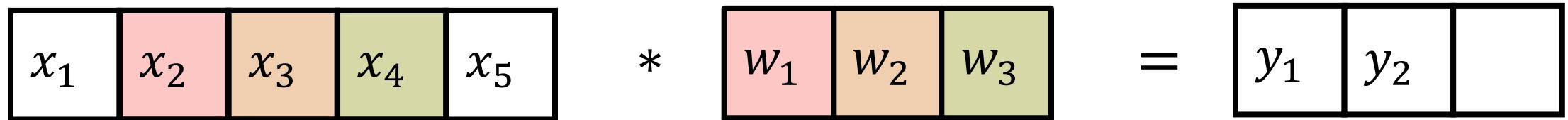
$$y_1 = w_1 x_1 + w_2 x_2 + w_3 x_3$$



Convolutional operation (2D)

The output computation now only depends on a subset of inputs:

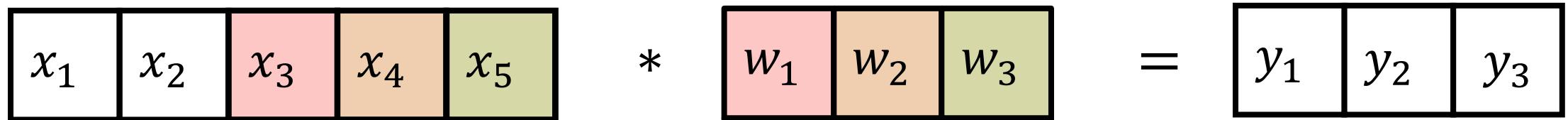
$$y_2 = w_1 x_2 + w_2 x_3 + w_3 x_4$$



Convolutional operation (2D)

The output computation now only depends on a subset of inputs:

$$y_3 = w_1 x_3 + w_2 x_4 + w_3 x_5$$



Translation equivariance

Translation equivariance is a property of Convolutional Neural Networks (CNNs) where the output of a CNN remains the same after an object within an image is translated (moved) by a fixed amount.

This means that if you translate an input image by a fixed number of pixels, the output of the CNN will be the same, up to the same translation.

→ An example of this property is seen in image classification tasks, where the presence of an object in an image remains the same regardless of its location within the image.

CNNs are **translation equivariant** because the same kernel is used to scan the entire image and compute the activation map: when the input image is translated, the same kernel is used to compute the activation map, leading to the same output, up to the same translation.



Deep Learning for Time Series – Convolutional Models

Convolutional-based Neural Networks (CNNs)



CNNs for time series

CNNs can be used to process sequential data, in place of RNNs.

- CNNs look forward, while RNN models only learn from data before the timestep it needs to predict.
 - CNNs (with shuffling) can see data from a broader perspective.
 - RNNs can learn causality

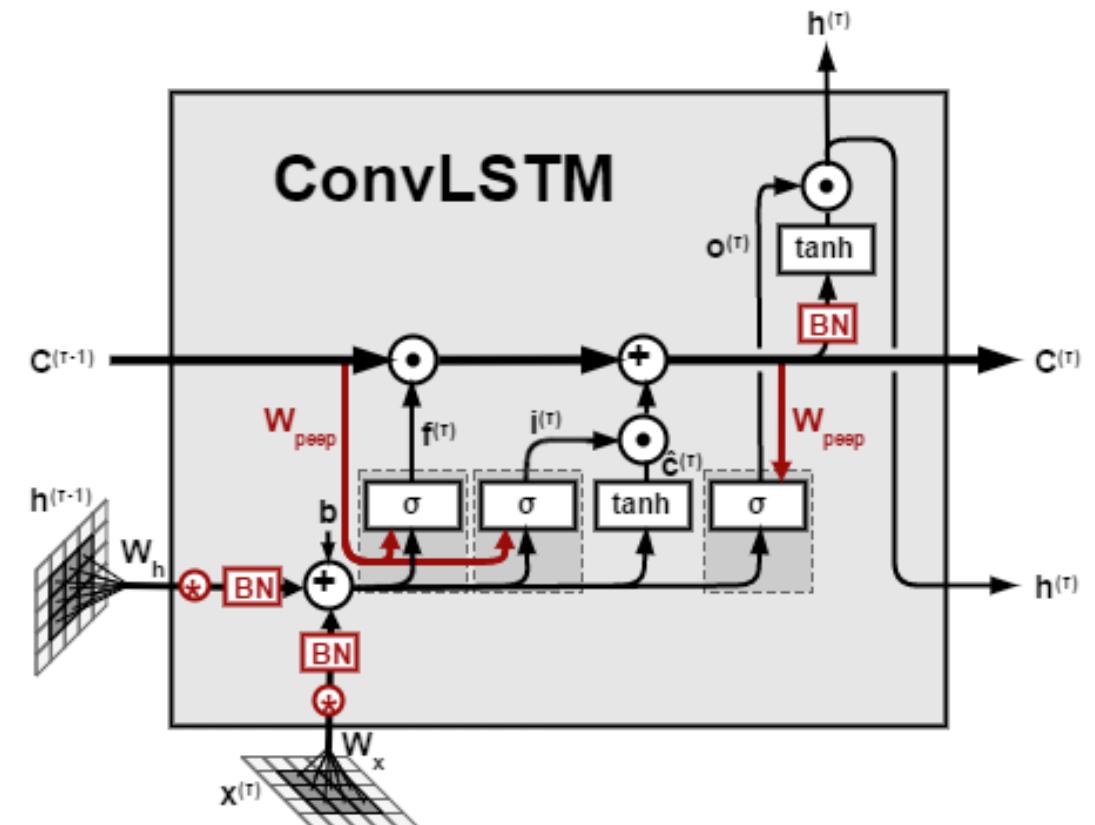
Definition. Causality is the influence that an event (cause) contributes to a successive event (effect).

A difference with conditional statements is that causality require the antecedent to precede or coincide with the consequent in time.

ConvLSTM

The ConvLSTM is a recurrent layer, just as like as LSTM, but the matrix multiplications within the LSTM cell are replaced by convolutional operations.

- Suited for video processing.
- It keeps the dimensionality of the input, e.g., 3D for a single-channel video.



ConvLSTM

The ConvLSTM is defined by the following equations:

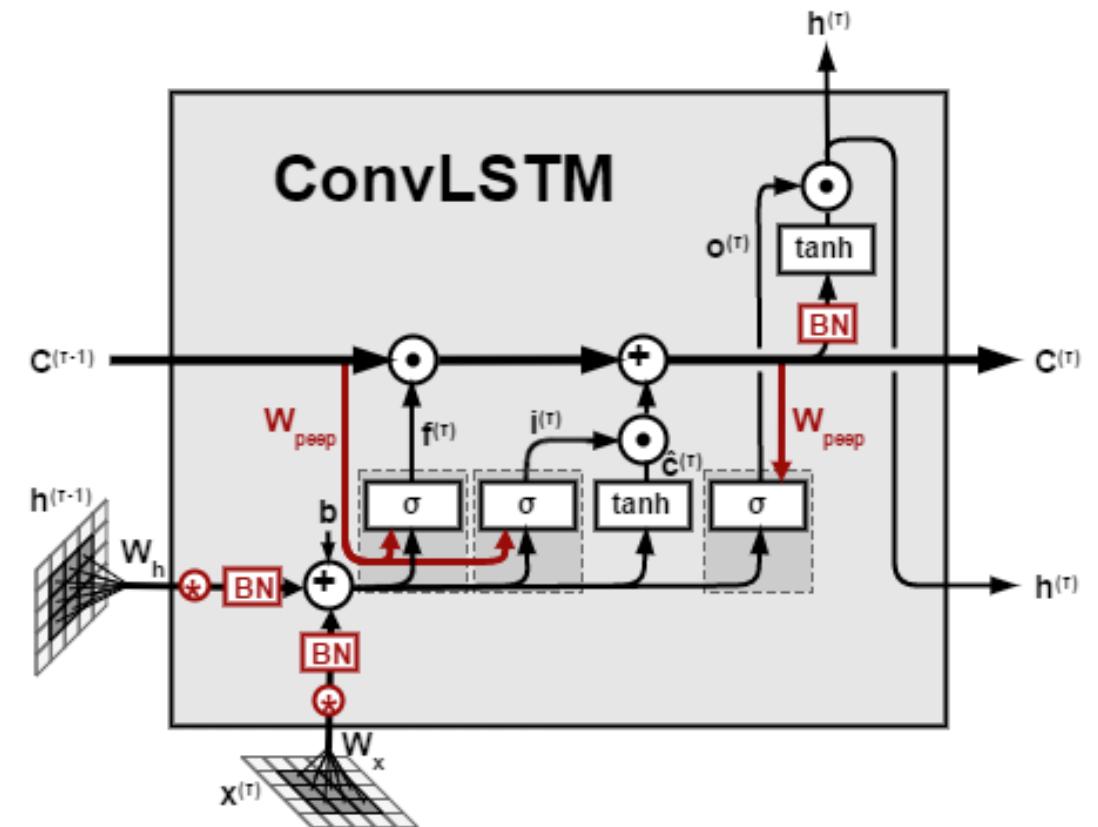
$$i_t = \sigma(W_{xi} * X_t + W_{hi} * H_{t-1} + W_{ci} \odot C_{t-1} + b_i)$$

$$f_t = \sigma(W_{xf} * X_t + W_{hf} * H_{t-1} + W_{cf} \odot C_{t-1} + b_f)$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \tanh(W_{xc} * X_t + W_{hc} * H_{t-1} + b_c)$$

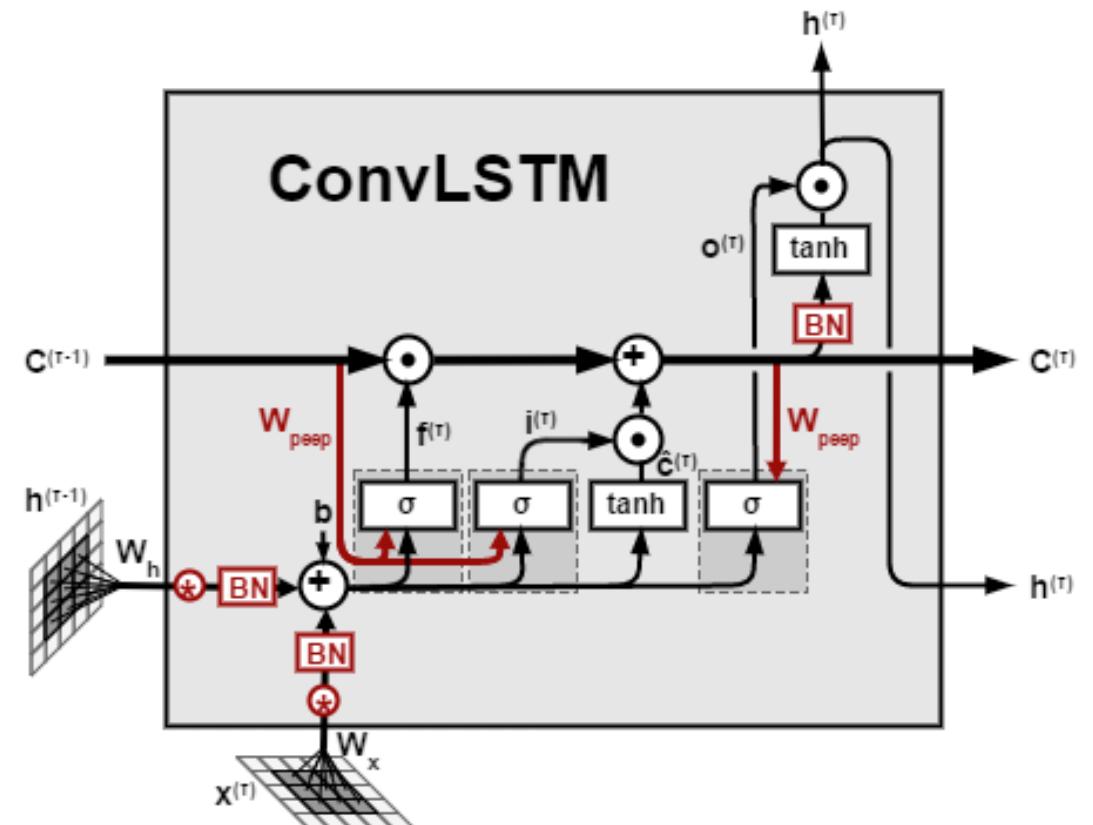
$$o_t = \sigma(W_{xo} * X_t + W_{ho} * H_{t-1} + W_{co} \odot C_t + b_o)$$

$$H_t = o_t \odot \tanh(C_t)$$



ConvLSTM

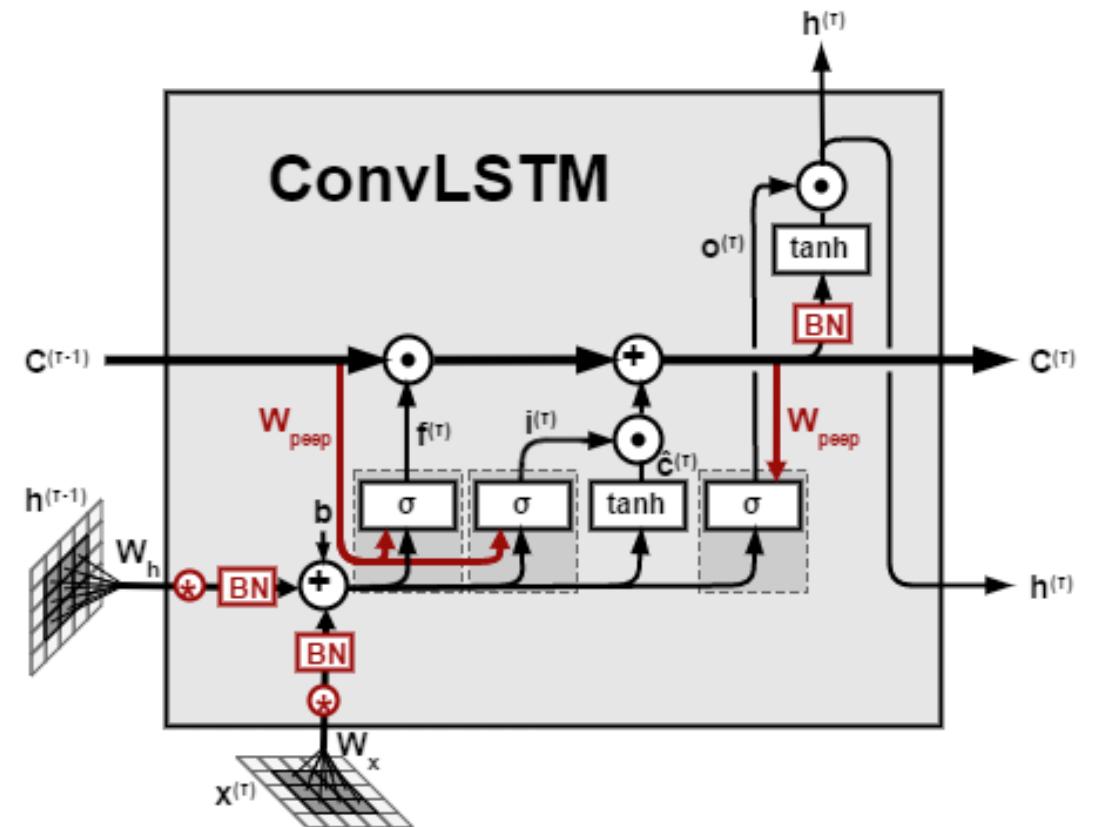
- ConvLSTM can be confused with CNN+LSTM models, i.e., a sequential combination of the two architectures which first processes an input frame through a CNN to create a flatten representation which is then fed as input to the LSTM.



ConvLSTM

ConvLSTM are successfully applied to different tasks:

- Gesture recognition [1]
- Precipitation nowcasting [2]
- Medical image segmentation [3]
- Video salient object detection [4]
- ...



[1] "Attention in Convolutional LSTM for Gesture Recognition", Zhang et al.

[2] "Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting", Shi et al.

[3] "Multi-level Context Gating of Embedded Collective Knowledge for Medical Image Segmentation", Asadi-Aghbolaghi et al.

[4] "Pyramid Dilated Deeper ConvLSTM for Video Salient Object Detection", Song et al.

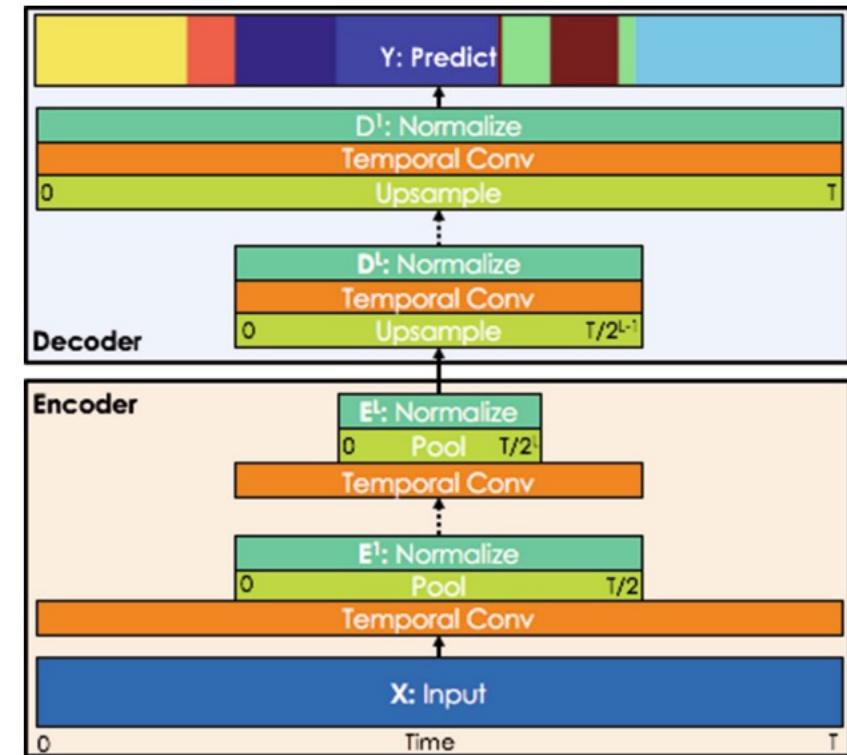
Temporal Convolutional Networks (TCNs)

Temporal Convolutional Networks (TCNs) were introduced in 2016 for video-based action segmentation [5].

It is a variation of CNN for better modeling of sequencing tasks.

The TCN provides a unified approach to capture both:

1. Low-level features encoding spatio-temporal information, and
2. High-level temporal information



[5] “Temporal convolutional networks: A unified approach to action segmentation”, Lea et al.

Temporal Convolutional Networks (TCNs)

Temporal Convolutional Networks (TCNs) are based on an encoder-decoder framework.

- It takes as input a sequence of any length and output a sequence of the same length.
- The causal convolution (or, temporal convolution) main characteristic is that the output at time t is only convolved with the observation until time t .
→ There is no information leakage from the future

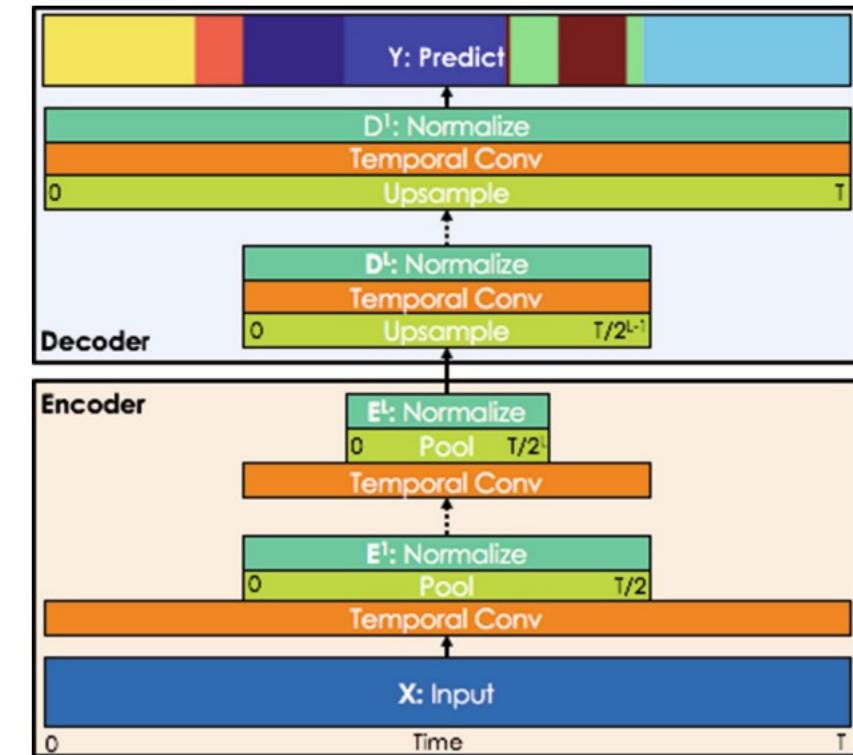


Image from the original paper: "Temporal convolutional networks: A unified approach to action segmentation", Lea et al.

Temporal Convolutional Networks (TCNs)

The causal convolution is best suited to model causality in the data.

- For images it can be implemented by “masked convolutions”, i.e., a tensor mask is applied before the actual convolution takes place.
- For 1D data, e.g., audio processing, it can be more easily implemented by shifting the output of a normal convolution by a few timesteps.

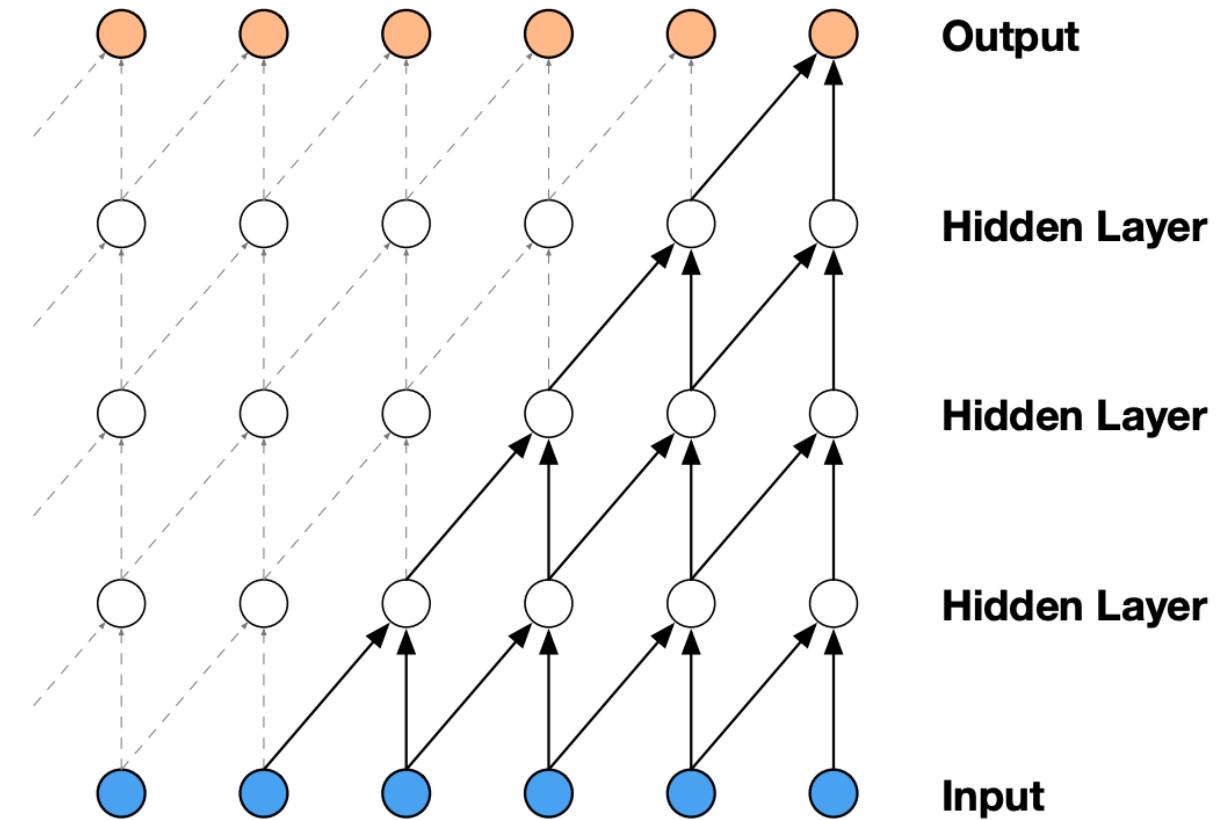
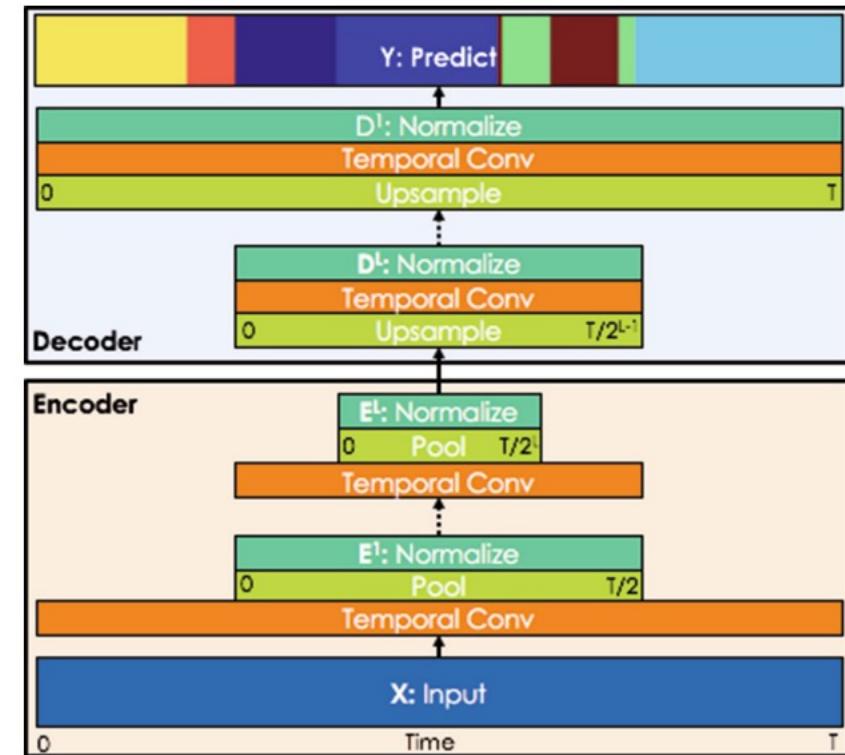


Image from the original paper: “Temporal convolutional networks: A unified approach to action segmentation”, Lea et al.

Temporal Convolutional Networks (TCNs)

Among a multitude of applications, TCNs have been successfully applied to:

- Weather prediction task [6]
- Sound event localization [7]
- Action segmentation [8]
- ...



[6] “Temporal convolutional networks for the Advance prediction of enSo”, Yan, Jining, et al.

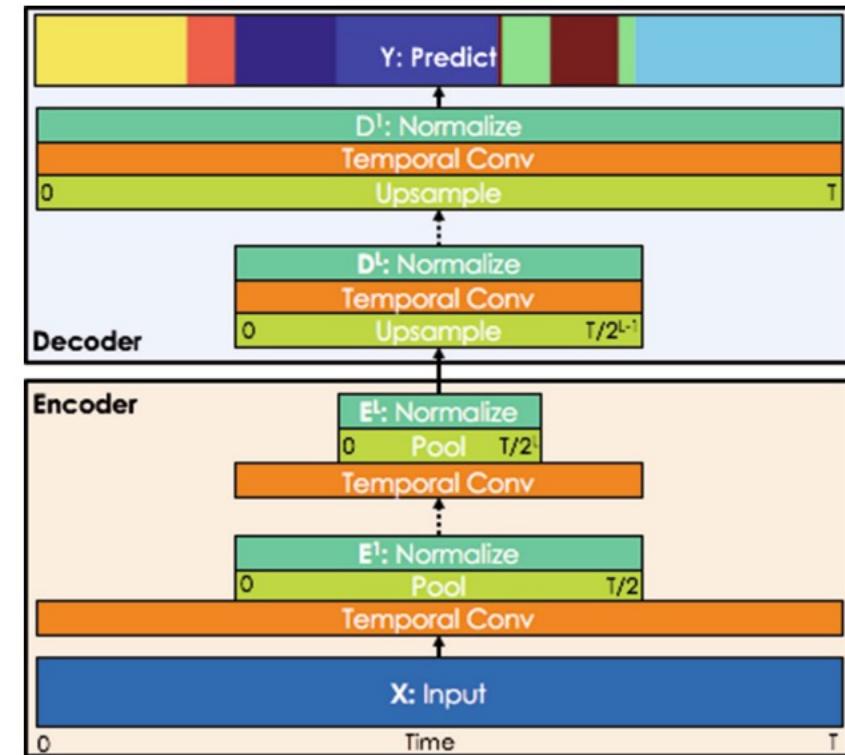
[7] “SELD-TCN: Sound Event Localization & Detection via Temporal Convolutional Networks.”, Guirguis et al.

[8] “Temporal convolutional networks: A unified approach to action segmentation.”, Colin et al.

Temporal Convolutional Networks (TCNs)

Advantages of TCNs:

- TCNs exhibit longer memory than RNNs, given the same capacity
- Parallelism of training



[6] “Temporal convolutional networks for the Advance prediction of enSo”, Yan, Jining, et al.

[7] “SELD-TCN: Sound Event Localization & Detection via Temporal Convolutional Networks.”, Guirguis et al.

[8] “Temporal convolutional networks: A unified approach to action segmentation.”, Colin et al.



Deep Learning for Time Series – Convolutional Models

Recap



In this lecture

- Convolutional neural networks (CNNs)
 - The convolution operation
 - Pooling
 - Padding
 - 1D and 2D convolutions
- Convolutional-based architectures
 - ConvLSTM
 - TCN

Critical comparison

Recurrent-based, convolutional-based, mixed

	FF-NN	RNN	CNN
Data	Tabular	Sequential	Grids
Weight-sharing	No	Yes	Yes
Recurrent connections	No	Yes	Yes
Equivariant	No	No	Yes
Parall. comp.	Yes	No	Yes
Vanishing grad.	Yes	Yes	Yes
Spatial relationships	No	No	Yes
Causality	No	Yes	No



Machine Learning for Time Series

(MLTS or MLTS-Deluxe Lectures)

Dr. Dario Zanca

Machine Learning and Data Analytics (MaD) Lab
Friedrich-Alexander-Universität Erlangen-Nürnberg
25.10.2022

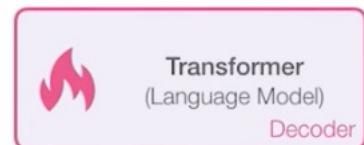
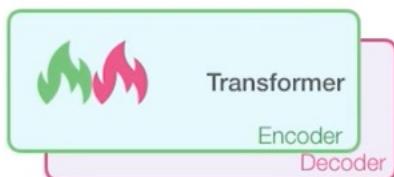
- Time series fundamentals and definitions (2 lectures)
- Bayesian Inference (1 lecture)
- Gaussian processes (2 lectures)
- State space models (2 lectures)
- Autoregressive models (1 lecture)
- Data mining on time series (1 lecture)
- Deep learning on time series (4 lectures) ←
- Domain adaptation (1 lecture)

Recap: Recurrent neural networks

RNN / LSTM limitations:

- Non-parallelism → Long training time
- Difficulties with long sequences
 - Large memory usage
 - Difficult to train (vanishing/exploding gradients)
 - Hard to learn long-term dependencies (mitigated by LSTMs)

Motivation



- Transformers *perceive* the entire sequence at the same time.
- They are based on the seq2seq concept, i.e., transforming sequences into other sequences.
- State of the art in many NLP tasks.

In this lecture...

- Attention models
- The transformers architecture



Deep Learning for Time Series – Attention models

Attention models



Sequence-to-sequence models

Sequence-to-sequence (seq2seq) models aim at transforming an input sequence to an output sequence

- E.g., machine translation.

Seq2seq models generally have an encoder-decoder architecture, composed by:

- An encoder that processes the input sequence and compress the information into a context vector (also said embedding).
- A decoder that processes the context vector and produces the transformed output.

Sequence-to-sequence models

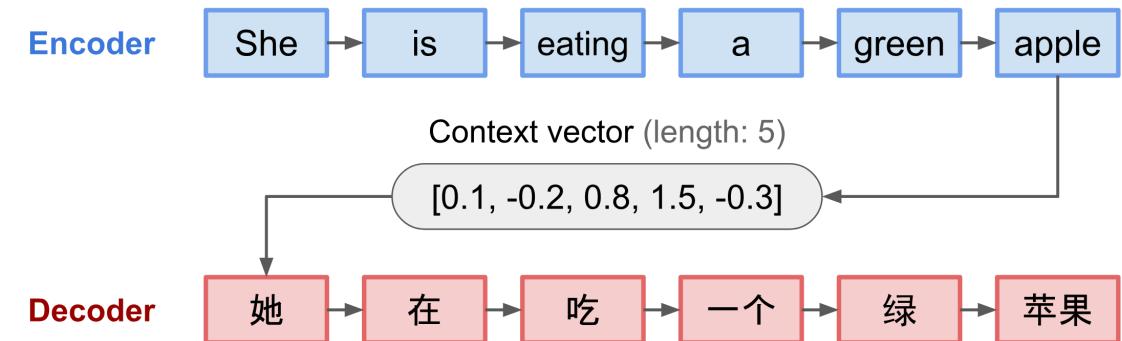
Disadvantages of the fixed length context vector design are:

- Incapability of remembering long sequences
- Too complex dynamics to be encoded in the hidden state

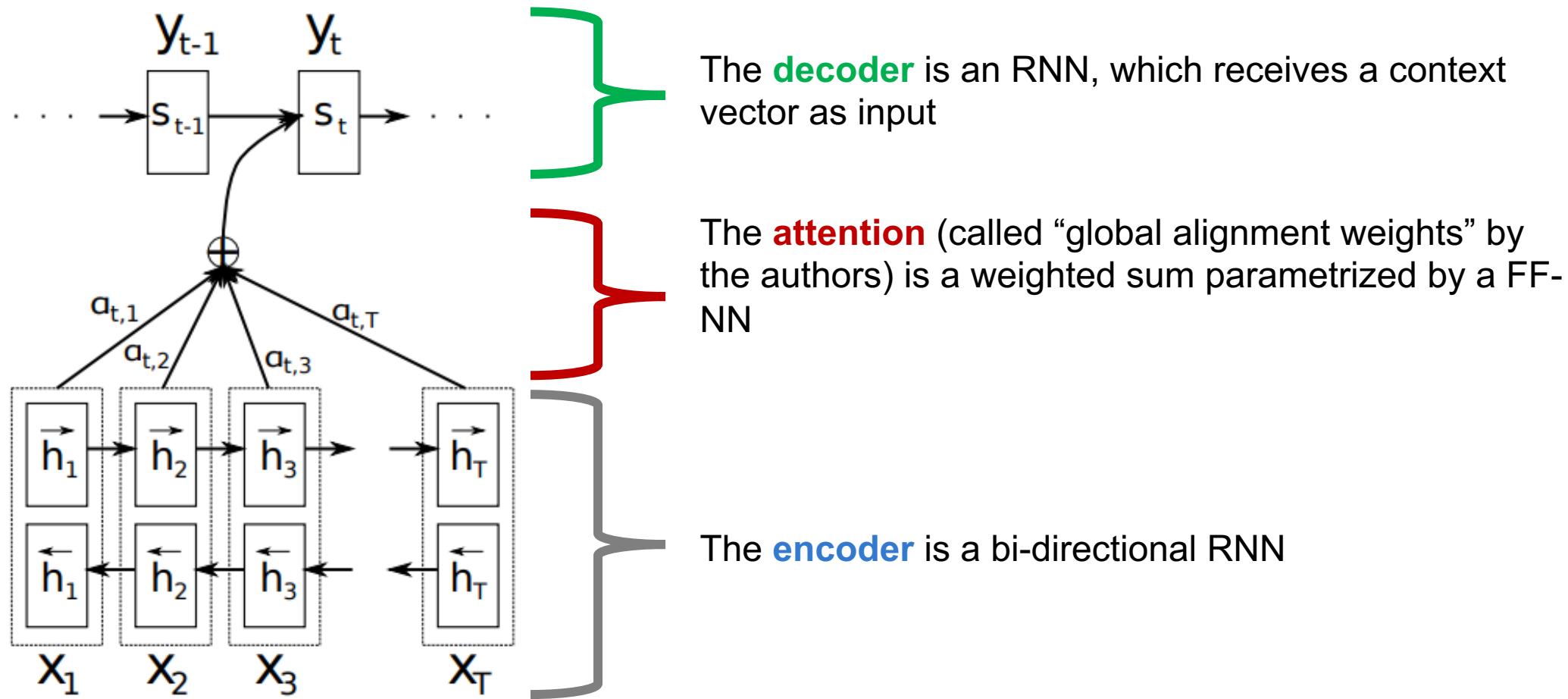
→ Attention mechanisms are proposed to solve this problem.

- Originated for machine translation [1]

[1] “Neural machine translation by jointly learning to align and translate”, Bahdanau et al.
Image from: <https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html>



Attention mechanism



[1] "Neural machine translation by jointly learning to align and translate", Bahdanau et al.

Attention mechanism: formalization

Let x be the input sequence of length n , and y the output sequence of length m .

Let $h_i = [\vec{h}_i, \hat{h}_i]$ be the encoder state, given by the concatenation of the forward and backward hidden states of the bidirectional RNN.

Let denote with s_t the decoder state, for the output at position t . Then, the context vector is defined as the sum of the encoder states, weighted by the alignment scores, i.e.,

$$c_t = \sum_{i=1}^n a_{t,i} h_i$$

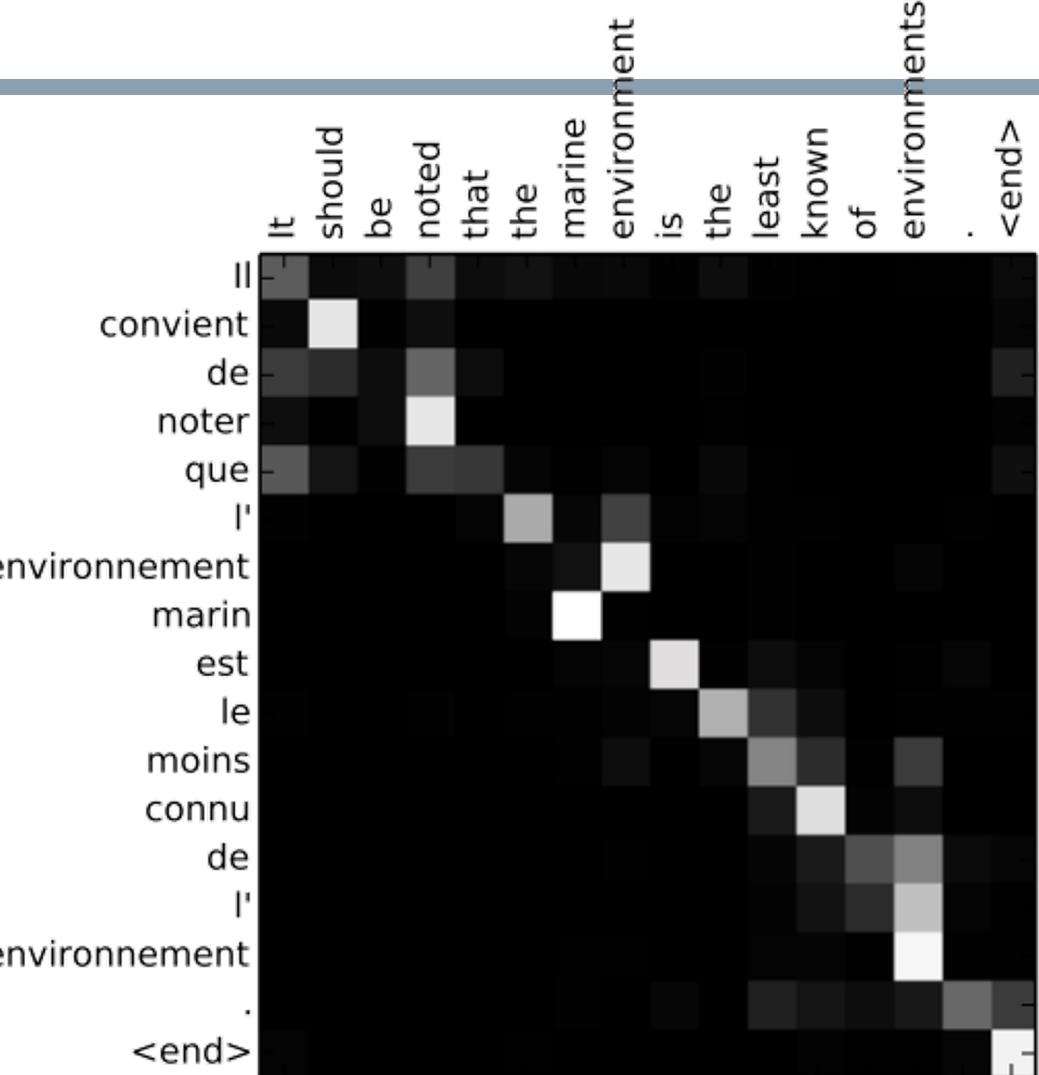
where $a_{t,i} = \text{softmax}(\text{score}(s_{t-1}, h_i))$, and $\text{score}(s_{t-1}, h_i) = v_a \tanh(W_a[s_{t-1}; h_i])$.

[1] "Neural machine translation by jointly learning to align and translate", Bahdanau et al.

Attention mechanism: example

In the example on the side, the x-axis represents the source sentence and the y-axis the generated sentence.

Every pixel (i, j) , at the i -th row and j -th column, indicates the weight the j -th source word embedding when generating the i -th word.



Attention mechanism

Name	Alignment score function
Content-base attention	$\text{score}(s_t, \mathbf{h}_i) = \text{cosine}[s_t, \mathbf{h}_i]$
Additive(*)	$\text{score}(s_t, \mathbf{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[s_t; \mathbf{h}_i])$
Location-Base	$\alpha_{t,i} = \text{softmax}(\mathbf{W}_a s_t)$ Note: This simplifies the softmax alignment to only depend on the target position.
General	$\text{score}(s_t, \mathbf{h}_i) = s_t^\top \mathbf{W}_a \mathbf{h}_i$ where \mathbf{W}_a is a trainable weight matrix in the attention layer.
Dot-Product	$\text{score}(s_t, \mathbf{h}_i) = s_t^\top \mathbf{h}_i$
Scaled Dot-Product(^)	$\text{score}(s_t, \mathbf{h}_i) = \frac{s_t^\top \mathbf{h}_i}{\sqrt{n}}$ Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state.

Table from: <https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.htm>

Attention mechanism

Attention mechanisms can be characterised according to their function or design as:

- **Self-attention**
- **Soft/Hard attention**
- **Global/Local attention**

Global / local attention

Self-attention, also said intra-attention, is an attention mechanisms which relates different positions of an input sequence to generate a representation of the same sequence.

The FBI is chasing a criminal on the run .
The **FBI** is chasing a criminal on the run .
The **FBI** **is** chasing a criminal on the run .
The **FBI** **is** **chasing** a criminal on the run .
The **FBI** **is** **chasing** **a** criminal on the run .
The **FBI** **is** **chasing** **a** **criminal** on the run .
The **FBI** **is** **chasing** **a** **criminal** **on** the run .
The **FBI** **is** **chasing** **a** **criminal** **on** **the** **run** .
The **FBI** **is** **chasing** **a** **criminal** **on** **the** **run** .
The **FBI** **is** **chasing** **a** **criminal** **on** **the** **run** .

Soft / hard attention

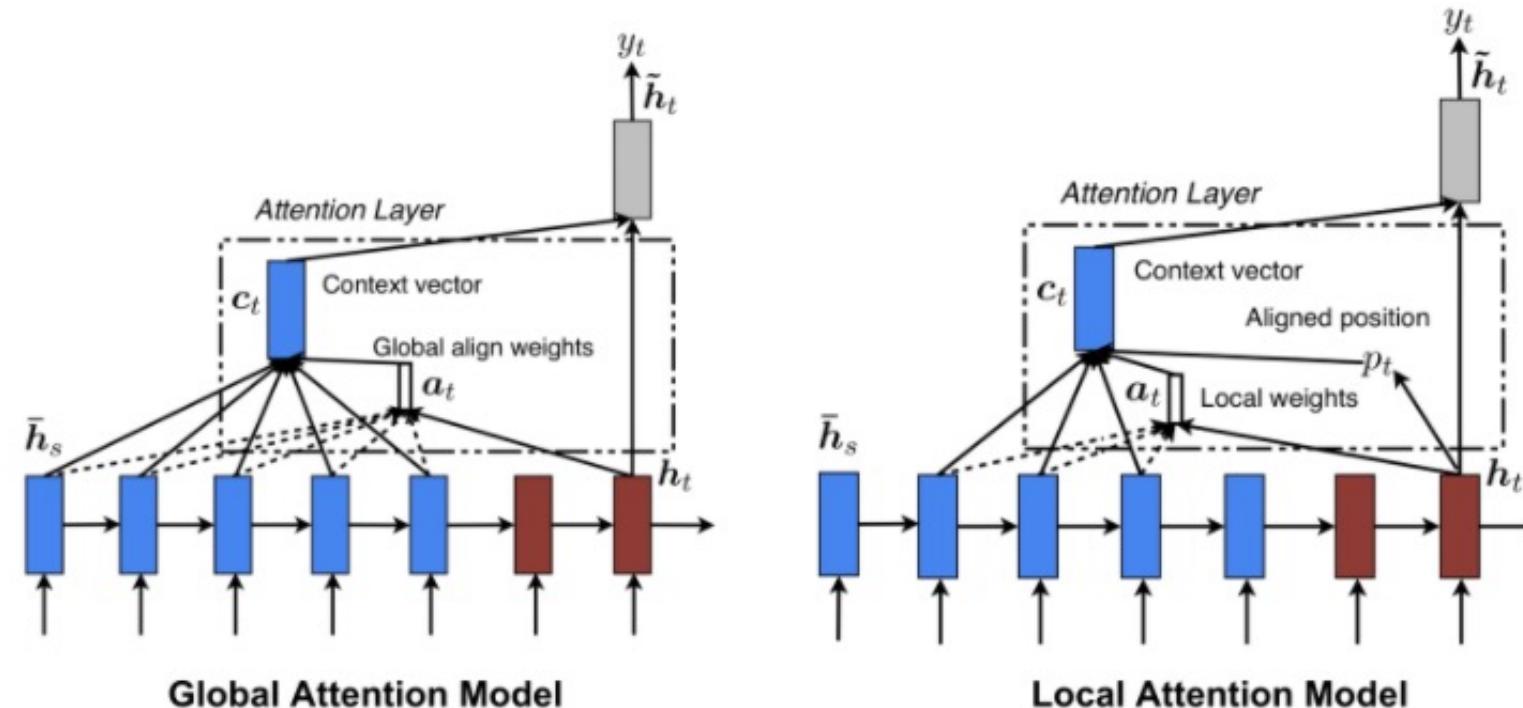
Soft attention applies learned weights to input parts, while **hard attention** select a single input part at the time.

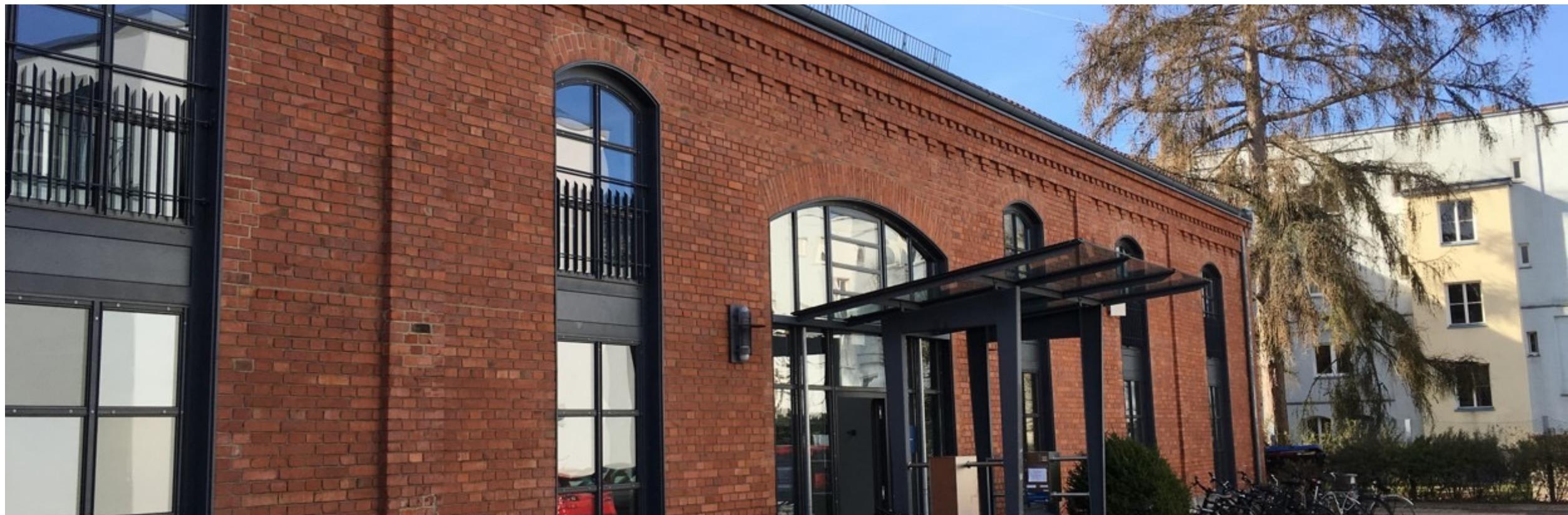


Image from: <https://glassboxmedicine.com/2019/08/10/learn-to-pay-attention-trainable-visual-attention-in-cnns/>

Global / local attention

In **global attention**, the context vector depends on the whole input sequence. The **local attention**, generates an input vector which depends only on a subset of the input sequence, corresponding to a window centered on the current position.





Deep Learning for Time Series – Attention models

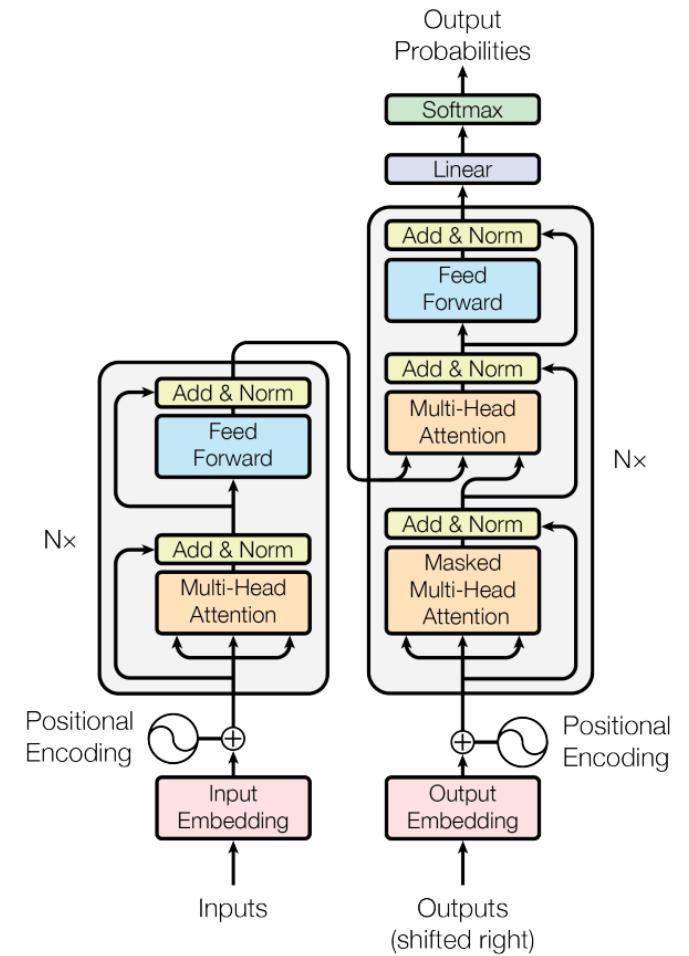
The Transformer architecture



The Transformer architecture

The Transformer architecture, introduced in 2017 [2]:

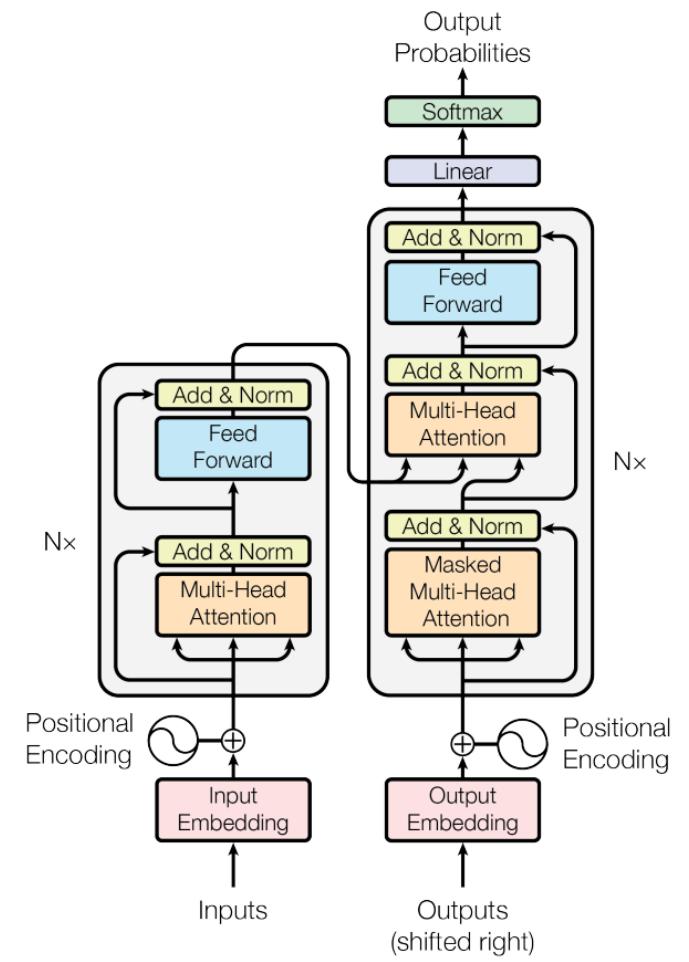
- Completely built on self-attention
 - Do not use sequence aligned recurrent architecture
- Replaced all the RNNs



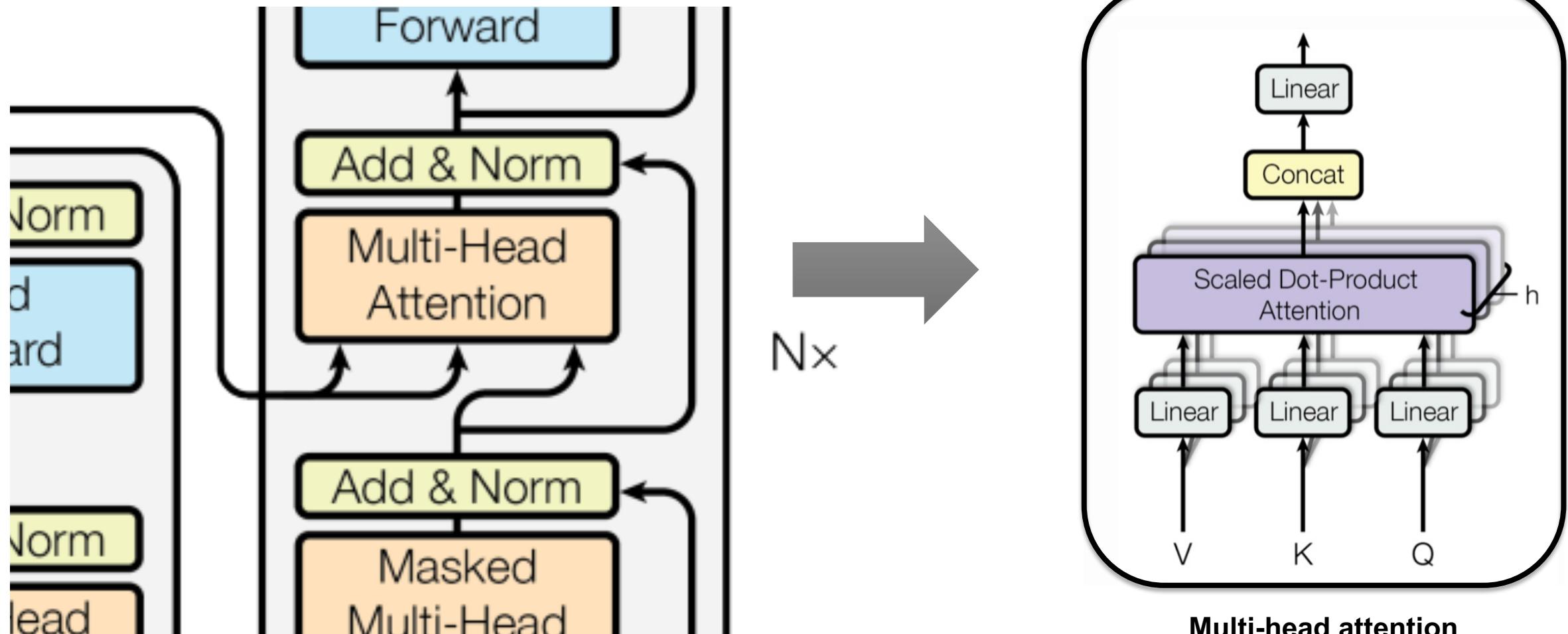
The Transformer architecture

The fundamental components of the transformer architectures are:

- Positional encoding
- Multi-head self attention
 - Based on K, V, and Q matrices
- An encoder-decoder architecture

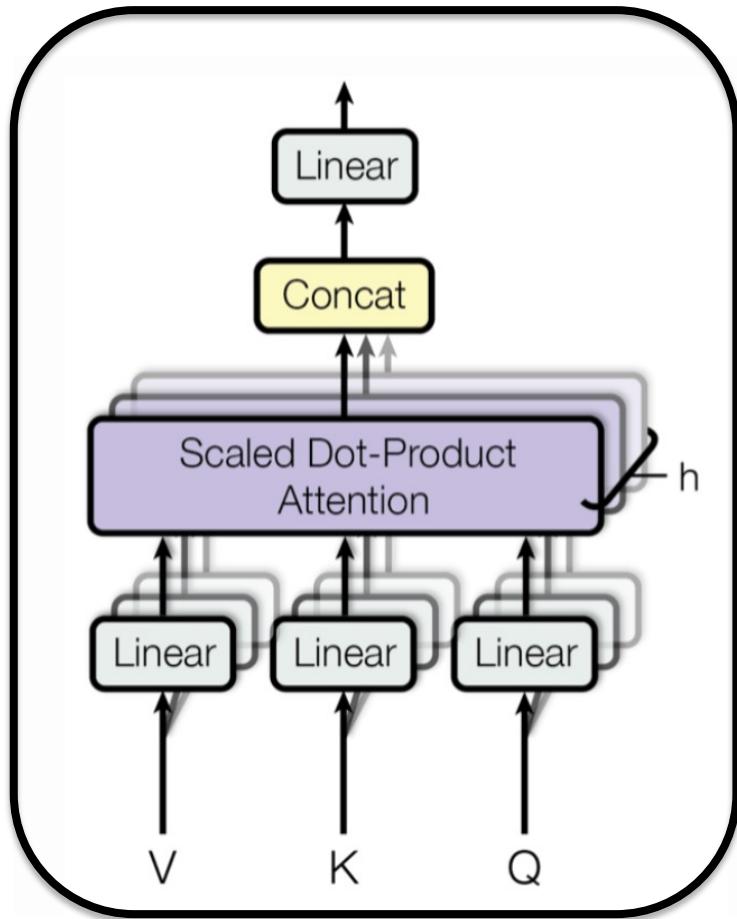


Transformer: multi-head attention



[2] "Attention is all you need", Vaswani, et al.

Transformer: multi-head self-attention



Multi-head attention

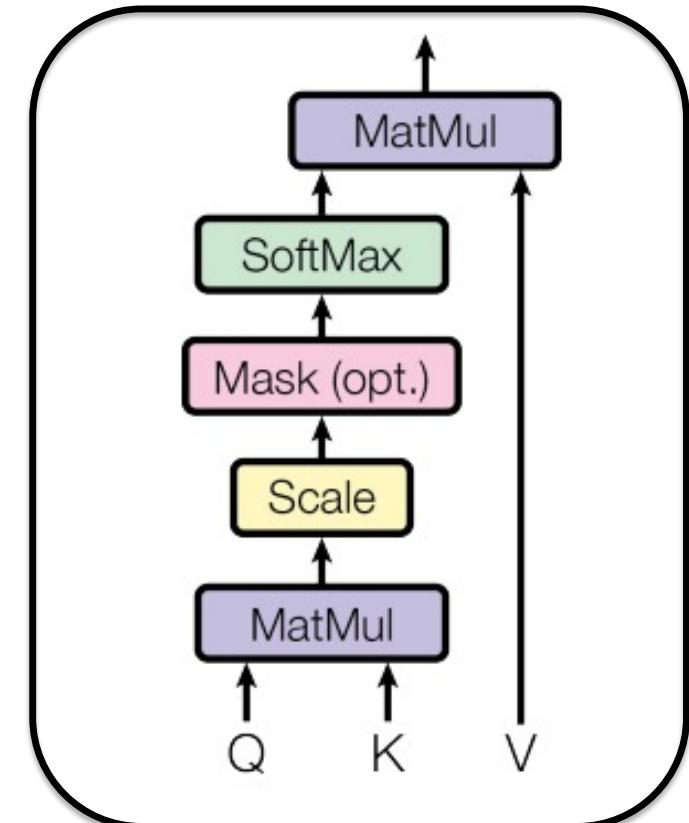
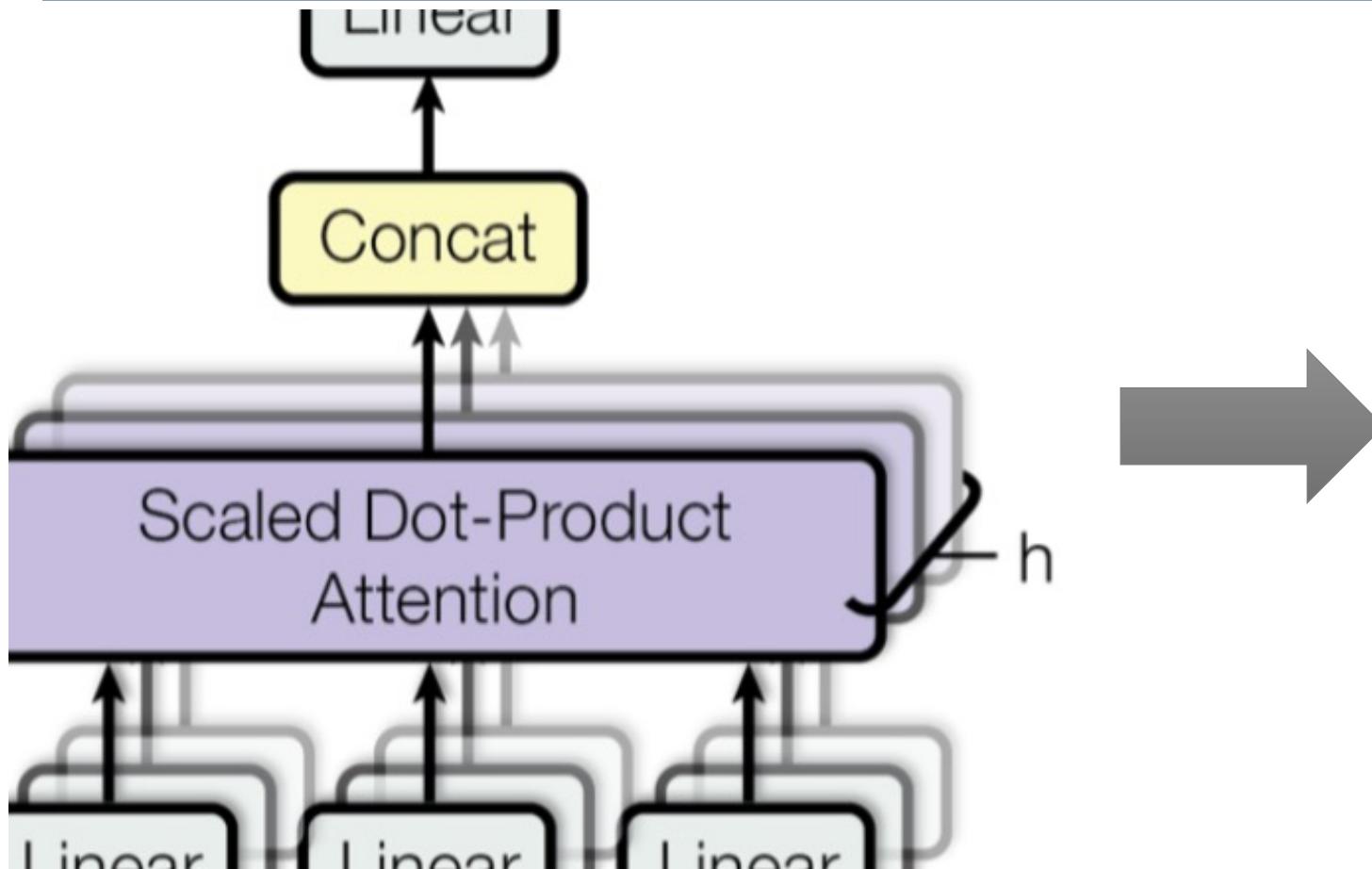
[2] “Attention is all you need”, Vaswani, et al.

The first step in calculating self-attention consists of calculating three vectors from each input vector (i.e., each element of the input sequence).

- Query, Q
- Key, K
- Value, V

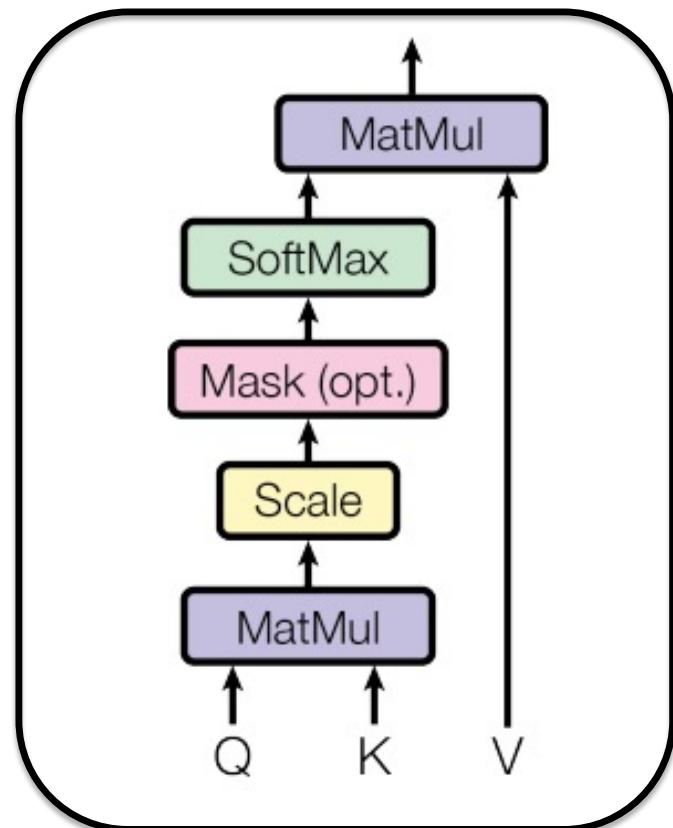
These vectors are calculated by multiplying the input vector by a corresponding matrix, resp., W^Q , W^K , and W^V .

Transformer: multi-head self-attention



Scaled Dot-Product Attention

Transformer: multi-head self-attention



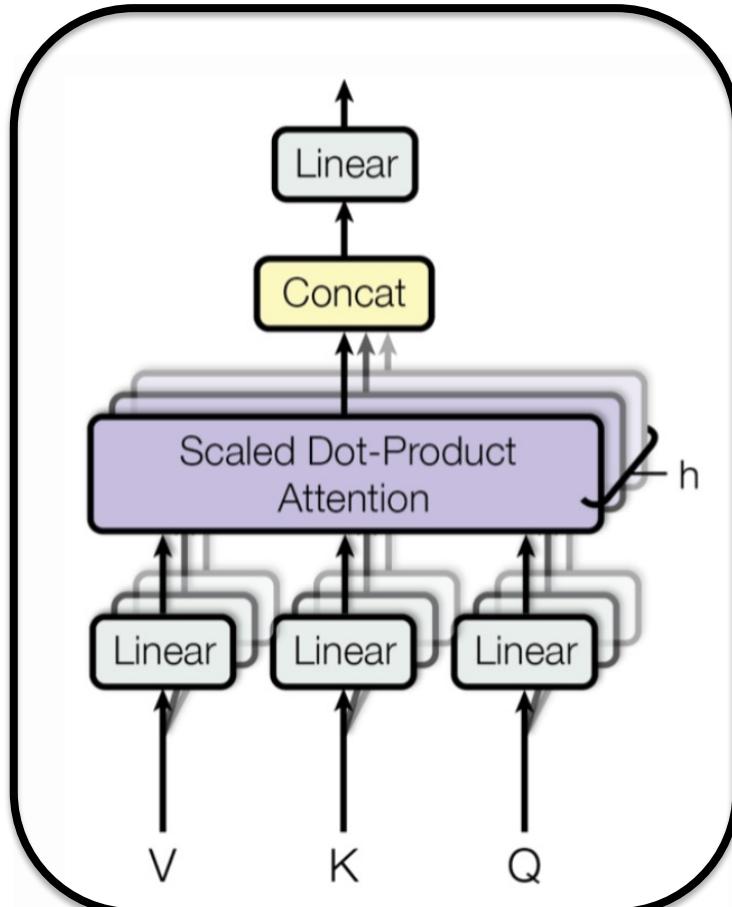
Scaled Dot-Product Attention

Then, the scaled dot-product attention is computed by

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V$$

where d_k is the dimension of queries and keys.

Transformer: multi-head self-attention



Multi-head attention

[2] "Attention is all you need", Vaswani, et al.

Instead of performing a single attention function with a d_{model} -dimensional keys, values and queries, it is beneficial to linearly project queries, keys and values with **different linear projections**.

Each of this projections is processed **in parallel** and then **concatenated**.

$$\text{MultiHead}(Q, K, V) = \text{concat}(\text{head}_1, \dots, \text{head}_h)$$

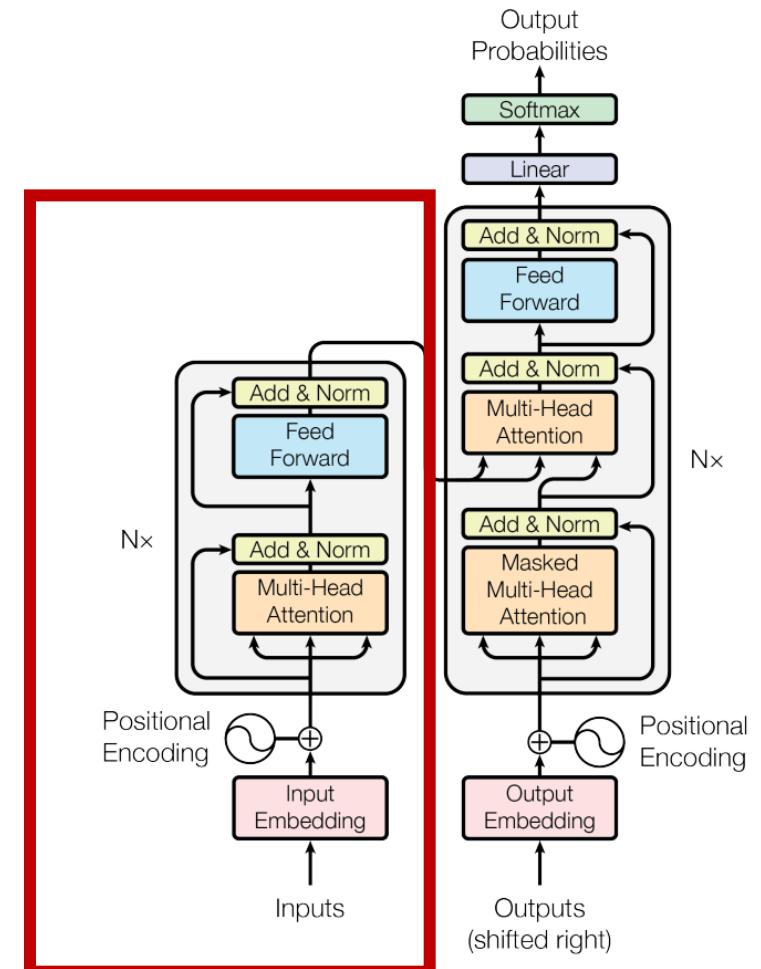
where each of the heads is a scaled dot-product attention, i.e.,

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Transformer: the encoder

The encoder generates an attention-based representation of the input sequence.

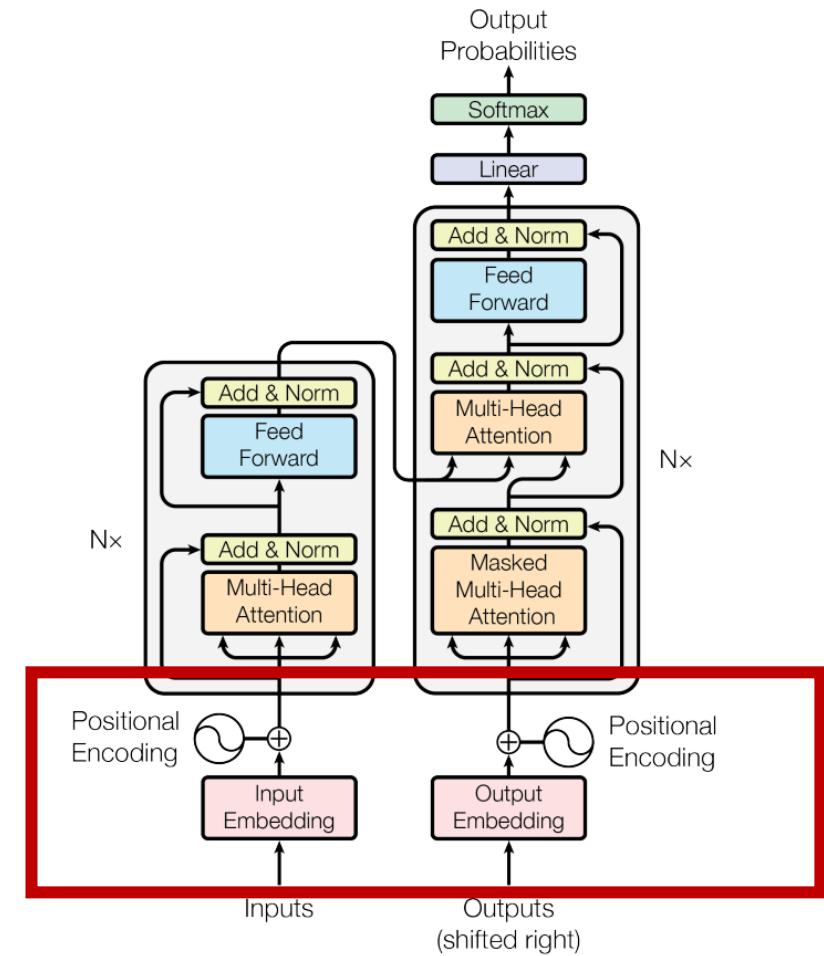
- Capability of locate a specific information from a (potentially) very large context.
- It is made of N_x identical layers,, each composed of
 - A multi-head attention layer
 - A positional FF-NN
 - A residual connection and layer normalization



Transformer: the positional encoding

Since the model contains no recurrence nor convolution, in order to make use of the order of the sequence, positional information is injected by mean of a **positional encoding**.

The positional encoding is **added to the input vectors** (for both the encoder and the decoder networks).



Transformer: the positional encoding

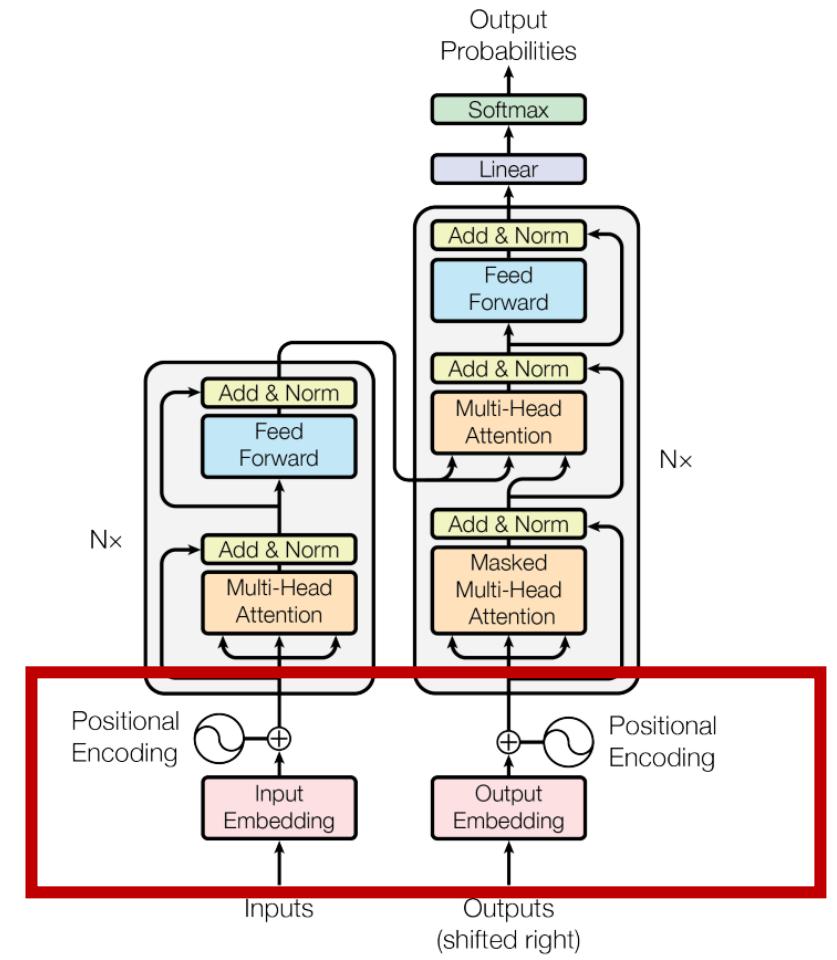
In the original implementations, the positional encoding is defined by using sine and cosine functions of different frequencies:

$$PE_{(pos, 2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

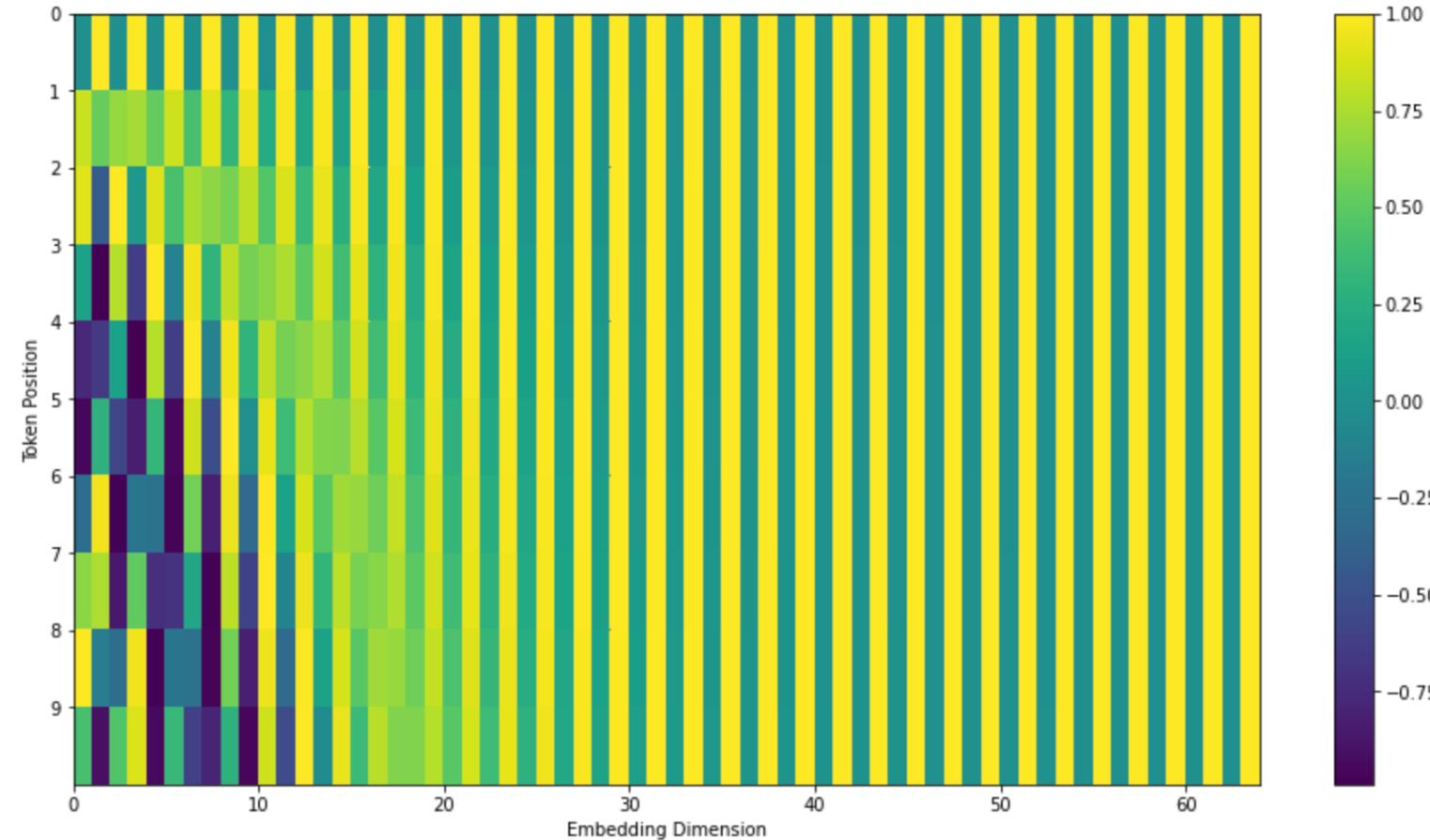
$$PE_{(pos, 2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

where pos is the position and i is the dimension.

[2] “Attention is all you need”, Vaswani, et al.

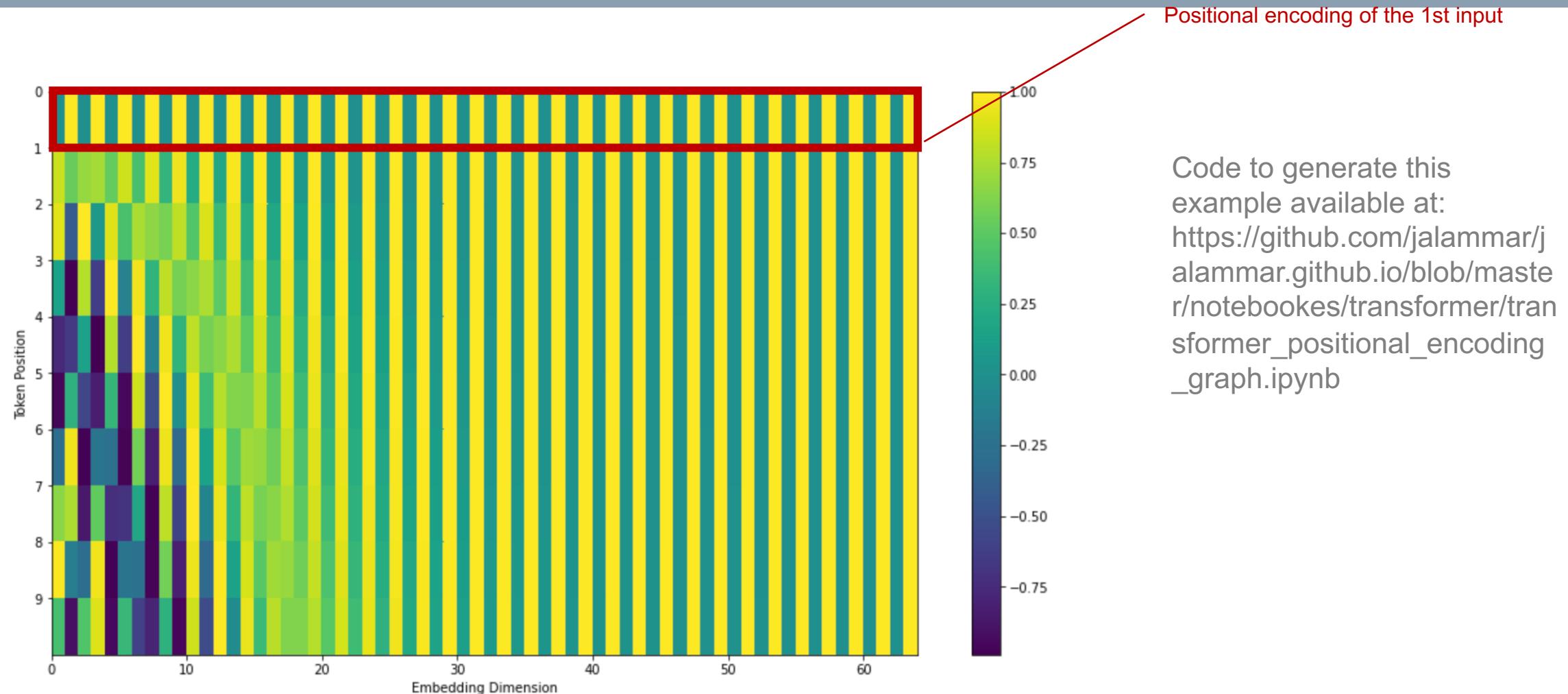


Transformer: the positional encoding

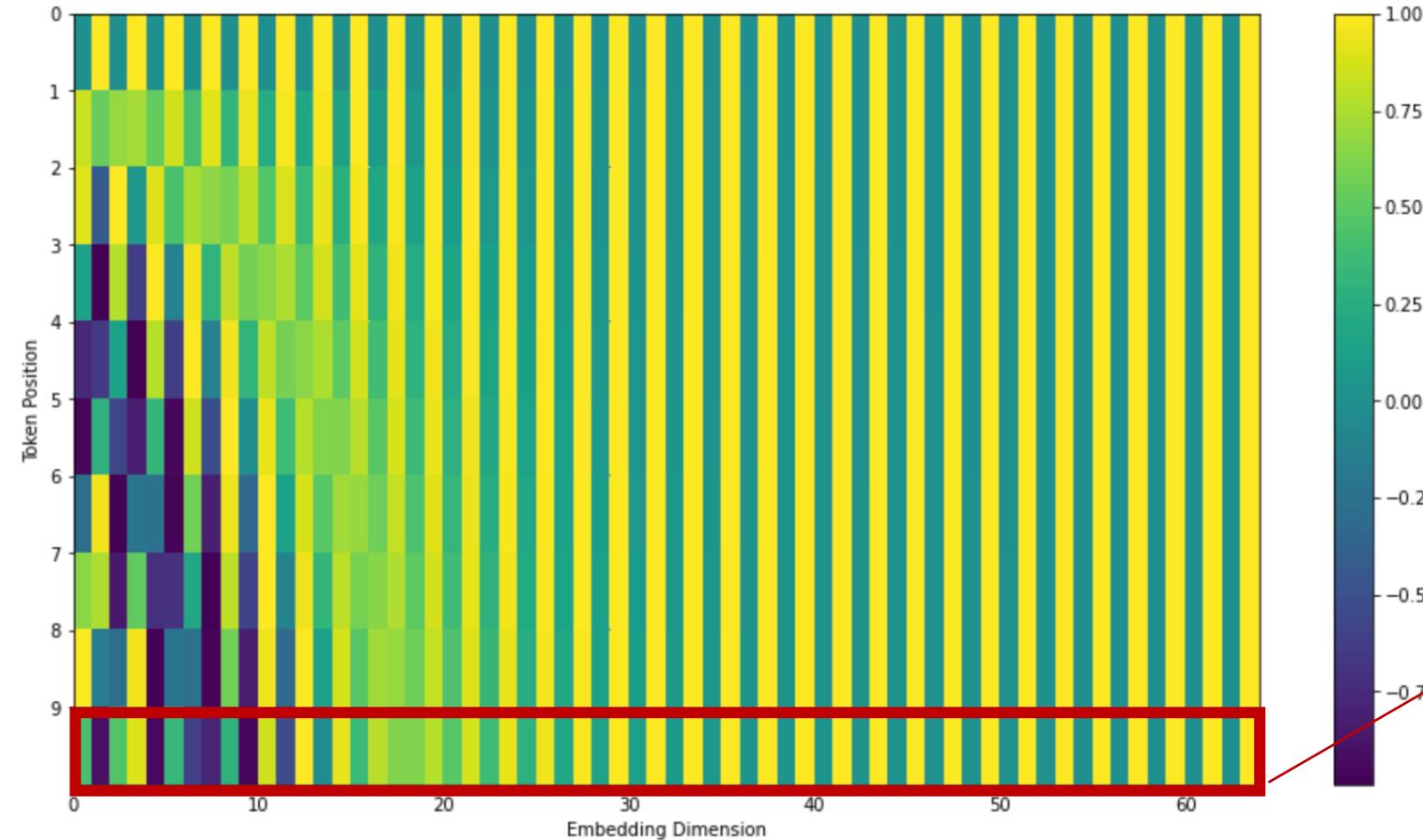


Code to generate this example available at:
https://github.com/jalammar/jalammar.github.io/blob/master/notebooks/transformer/transformer_positional_encoding_graph.ipynb

Transformer: the positional encoding



Transformer: the positional encoding



Code to generate this example available at:
https://github.com/jalammar/jalammar.github.io/blob/master/notebooks/transformer/transformer_positional_encoding_graph.ipynb

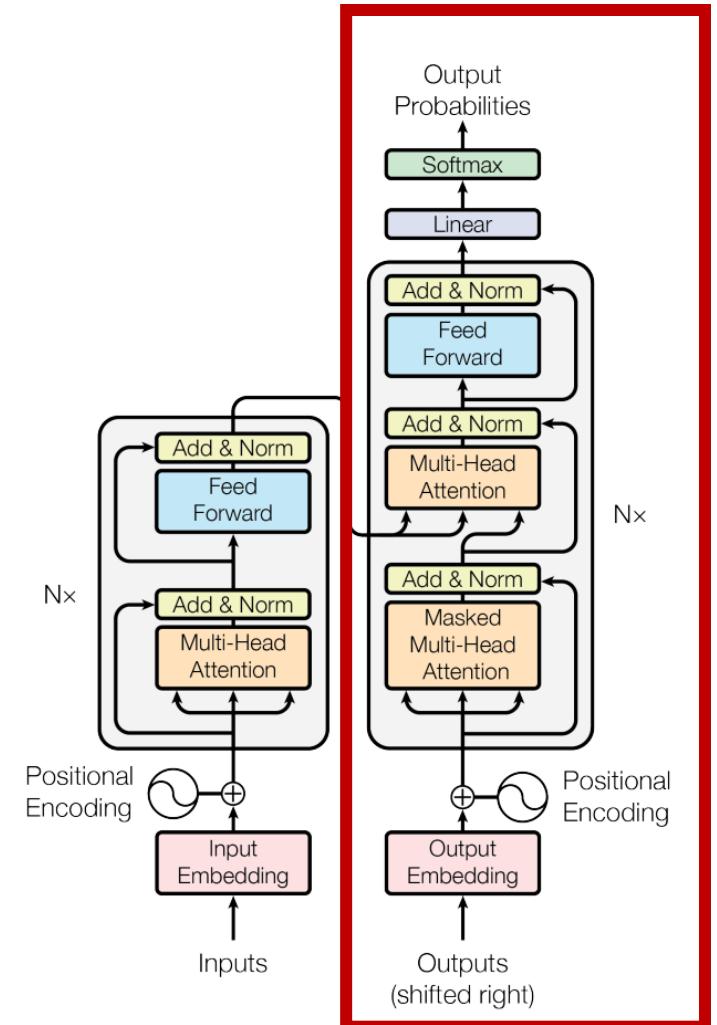
Positional encoding of the 10th input

Transformer: the decoder

The decoder network is able to retrieval information from the encoded representation.

- As the encoder, it is made of N_x identical layers,, each composed of
 - A multi-head attention layer
 - A positional FF-NN
 - A residual connection and layer normalization
- The first multi-head self-attention is properly masked to prevent information leakage from future positions.

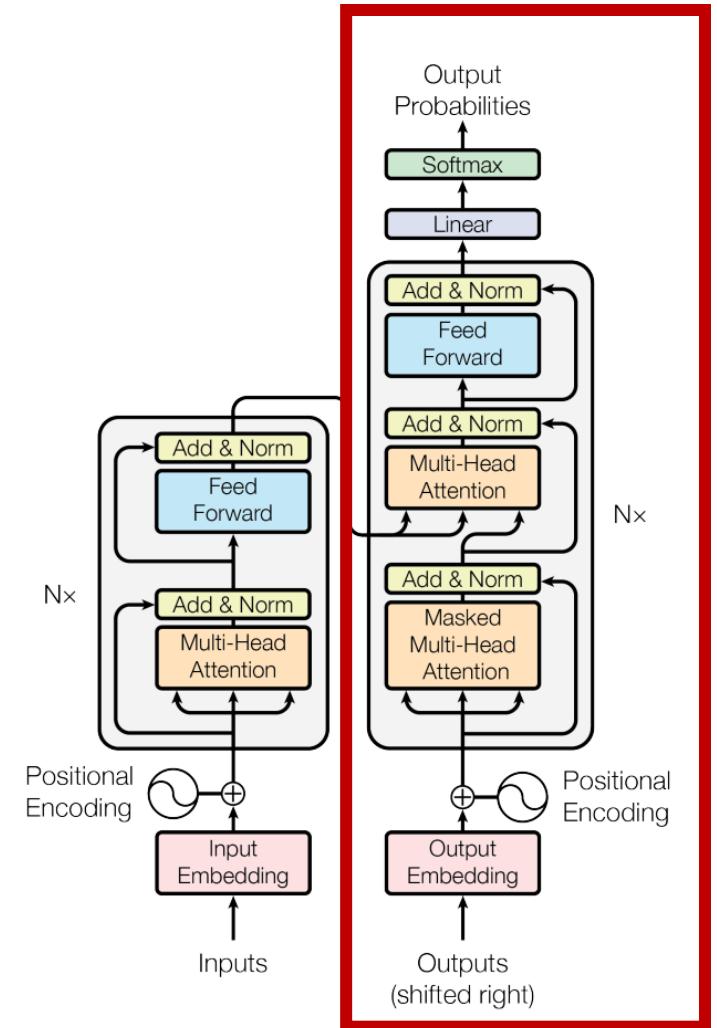
[2] “Attention is all you need”, Vaswani, et al.

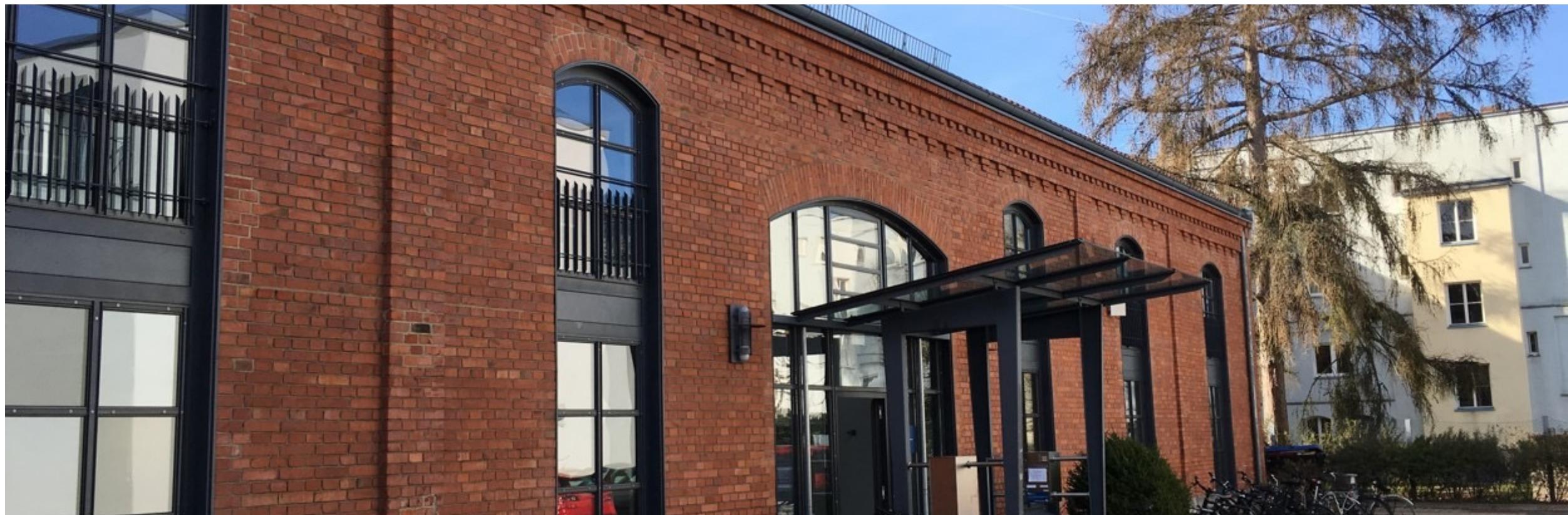


Transformer: the decoder

The deconding works as follows:

1. The output of the top encoder is transformed into a set of attention vectors K and V .
2. These are used by each decoder
→ It helps the decoder focus on appropriate parts
3. The process is repeated until a special symbol of <end> is generated.





Lecture title

Recap



In this lecture

- Attention models
 - Seq2Seq
 - Encoder-decoder
 - All hidden states are passed
 - Self-attention, soft/hard, local/global
- Transformers
 - Multi-head Self-attention
 - Positional encoding
 - Encoder and decoder networks

Transformers: pros and cons

Pros:

- Transformers can learn direct access to potentially very far input parts
- Many degree of freedom → Can learn complex dependecies
- Feed-forward model provides high performance on moder hardware

Cons:

- Quadratic time and memory complexity
- Many degree of freedom → Data hungry behavior during training
- Training is insidious
- Hard integration with other architecture, not easy learning rate policy, use custom data loader, ...
- Still limited research on time series data.

Other attention-based approaches

- Neural Turing Machines [3]
 - Couples NNs with external memory
 - Mimics reading and writing operations in Turing machines
 - Exploit content-based attention
- Pointer networks [4]
 - Solves problems where the outputs are positions in an input sequence
 - It applies attention over the input elements to pick one as the output at each decoder step.
- ...



Machine Learning for Time Series

(MLTS or MLTS-Deluxe Lectures)

Dr. Dario Zanca

Machine Learning and Data Analytics (MaD) Lab
Friedrich-Alexander-Universität Erlangen-Nürnberg
25.10.2022

- Time series fundamentals and definitions (2 lectures)
- Bayesian Inference (1 lecture)
- Gaussian processes (2 lectures)
- State space models (2 lectures)
- Autoregressive models (1 lecture)
- Data mining on time series (1 lecture)
- Deep learning on time series (4 lectures)
- Domain adaptation (1 lecture) ←

In this lecture...

- 1. Domain adaptation: overview**
- 2. Unsupervised domain adaptation**
- 3. Domain generalization (OOD generalization)**



Domain adaptation

Domain adaptation: overview



The typical setup we have had so far included a training set

$$\{(x_i^{train}, y_i^{train})\}_{i=1}^m \sim Q_{X,Y}$$

Where $x_i \in X$, $y_i \in Y$, and where $Q_{X,Y}$ denotes the distribution from the training examples are sampled from.

Again, typically we want to learn an optimal mapping f_θ , for which we solve:

$$\min_{\theta} \frac{1}{m} \sum_{i=1}^m L(f_\theta(x_i^{train}), y_i^{train}) \Rightarrow \theta^*$$

We, then, evaluate our model on a hold out test set

$$\{(x_i^{test}, y_i^{test})\}_{i=1}^{m'} \sim Q_{X,Y}$$

by computing a test error

$$\epsilon_{test} = \frac{1}{m} \sum_{i=1}^{m'} L(f_{\theta^*}(x_i^{test}), y_i^{test})$$

(and we aim at a small ϵ_{test}).

Summary:

1. $\{(x_i^{train}, y_i^{train})\}_{i=1}^m \sim Q_{X,Y}$
2. $\min_{\theta} \frac{1}{m} \sum_{i=1}^m L(f_{\theta}(x_i^{train}), y_i^{train}) \Rightarrow \theta^*$
3. $\{(x_i^{test}, y_i^{test})\}_{i=1}^{m'} \sim Q_{X,Y}$
4. $\epsilon_{test} = \frac{1}{m} \sum_{i=1}^{m'} L(f_{\theta^*}(x_i^{test}), y_i^{test})$

Key assumption is that both the training and test set come from the same distribution.

Is it a realistic assumption?

In practice, the training distribution and the test distribution are often not the same.

- We train an image classifier on a database of photos taken with a professional camera, and want our classifier to work on pictures taken with any smartphone camera.
- Training distribution ≠ Test distribution
- $Q_{X,Y} \neq P_{X,Y}$

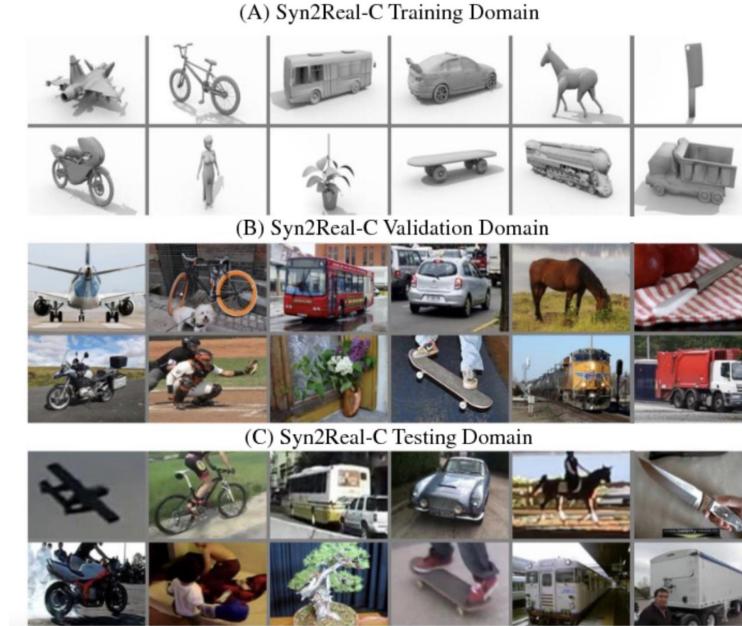
We introduce some terminology from the Domain Adaptation domain:

- **Source domain $Q_{X,Y}$.** The data distribution on which the model is trained using labeled examples.
→ photos taken with a professional camera.
- **Target domain $P_{X,Y}$.** A different, yet “related” distribution on which it is required to perform a similar task.
→ photos taken with a smartphone.
- **Domain shift.** It is the statistical difference between different domains.
→ statistical difference between $Q_{X,Y}$ and $P_{X,Y}$.

We introduce some terminology from the Domain Adaptation domain:

- **Source domain $Q_{X,Y}$.** The data distribution on which the model is trained using labeled examples.
→ photos taken with a professional camera.
- **Target domain $P_{X,Y}$.** A different, yet “related” distribution on which it is required to perform a similar task.
→ photos taken with a smartphone.
- **Domain shift.** It is the statistical difference between different domains.
→ statistical difference between $Q_{X,Y}$ and $P_{X,Y}$.

Domain adaptation examples



Train on synthetic samples and use with real samples.

Image from : Peng X. et al., "Syn2Real: A New Benchmark for Synthetic-to-Real Visual Domain Adaptation"



Same view from different seasons

Image from : Olid D. et al., "Single-View Place Recognition under Seasonal Changes"

Unsupervised domain adaptation

- Labeled samples for the source domain

$$Q_{X,Y} \sim \{(x_i^S, y_i^S)\}_{i=1}^{m_S} := (X^S, Y^S)$$

- Only unlabeled samples available for the target domain

$$P_X \sim \{x_i^T\}_{i=1}^{m_T} := X^T$$

Semi-supervised domain adaptation

- Labeled samples for the source domain

$$Q_{X,Y} \sim \{(x_i^S, y_i^S)\}_{i=1}^{m_S} := (X^S, Y^S)$$

- Unlabeled target samples + “Few” labeled target samples

Domain generalization

- Labeled samples for the multiple source domains $Q_{X,Y}^1 \sim \{(x_i^{S_1}, y_i^{S_1})\}_{i=1}^{m_{S_1}} := (X^{S_1}, Y^{S_1})$
 $Q_{X,Y}^2 \sim \dots$
 \dots
- **No** samples from the target domain available during training
- This problem is also called “out-of-distribution generalization”

Notice:

- We use both the samples from the source domain and from the target domain during training
- The target domain is different than what we use to call test set
- We need labelled samples from the target domain for testing, in all three scenarios



Domain adaptation

Unsupervised domain adaptation



Let's assume, for simplicity and without loss of generalization, that $m_S = m_T = m$, i.e.,

- Source domain: $(X^S, Y^S) = \{(x_i^S, y_i^S)\}_{i=1}^m \sim Q_{X,Y}$
- Target domain: $X^T = \{x_i^T\}_{i=1}^m \sim P_X$

The goal in unsupervised domain adaptation is that of, given a hypothesis class H , to pick a function $h \in H$ such that

$$\epsilon_T(h) = \mathbb{E}[L(h(x), y)]$$

with $(x, y) \sim P_{X,Y}$

Let's assume, for simplicity and without loss of generalization, that $m_S = m_T = m$, i.e.,

- Source domain: $(X^S, Y^S) = \{(x_i^S, y_i^S)\}_{i=1}^m \sim Q_{X,Y}$
- Target domain: $X^T = \{x_i^T\}_{i=1}^m \sim P_X$

The goal in unsupervised domain adaptation is that of, given a hypothesis class H , to pick a function $h \in H$ such that

$$\epsilon_T(h) = \mathbb{E}[L(h(x), y)]$$

with $(x, y) \sim P_{X,Y}$

- 1. Covariate shifts.** P and Q satisfy the covariate shift assumption if the conditional label distribution does not change between source and target distribution.

$$\forall x \in X, y \in \{0, 1\} \Rightarrow P(y | x) = Q(y | x)$$

- 2. Similarity of distributions.** Source and target (marginal) distribution should be similar.

$$Q_X \dots < = > \dots P_X$$

- 3. Small joint error.** If I “had” labeled samples, the joint error should be small.

$$\epsilon_{joint} = \min \left[\frac{1}{m} \sum_{i=1}^m L(h(x_i^S), y^S) + \frac{1}{m} \sum_{i=1}^m L(h(x_i^T), y^T) \right] \approx 0$$

H-divergence is defined as:

$$2 \sup_{h \in H} |p_{x \in Q_X}(h(x) = 1) - p_{x \in P_X}(h(x) = 1)| \triangleq d_H(Q_X, P_X)$$

Lemma. The H-divergence $d_H(Q_X, P_X)$ can be estimated by $m_S = m_T = m$ samples from source and target domains, $VC(H) = d$, with probability $1 - \delta$,

$$d_H(Q_X, P_X) \leq d_H(Q_X^{(m)}, P_X^{(m)}) + 4 \sqrt{\frac{d \log(2m) - \log(\frac{2}{\delta})}{m}}$$

Estimate H-divergence

Methods for estimating the H-divergence

The H-divergence can be computed by finding a classifier to separate source domain from target domain.

- Label all source samples as +1
- Label all target samples as 0
- Train a classifier to minimize the classification error:

$$\epsilon_{class} = \min_{h \in H} \left[\frac{1}{m} \sum_{i=1}^m 1(h(x_i^S) = 0) + \frac{1}{m} \sum_{i=1}^m 1(h(x_i^T) = 1) \right]$$

The classification loss is inversely proportional to the H-divergence,

$$\frac{1}{2} d_H(Q_X^{(m)}, P_X^{(m)}) = 1 - \epsilon_{class}$$

Definition

Definition. For the hypothesis class H , the symmetric difference hypothesis space $H\Delta H$ is the set of disagreements between any two hypothesis in H .

$$H\Delta H = \{g(x) = h(x) \oplus h'(x) | h, h' \in H\}$$

The following “main result” has inspired many practical methods in domain adaptation.

Main result. H is a hypothesis class with $VC(H) = d$. We are given unlabeled samples from the target $P_X^{(m)}$ and labeled samples from the sources $Q_{X,Y}^{(m)}$. With probability $1 - \delta$, for any $h \in H$,

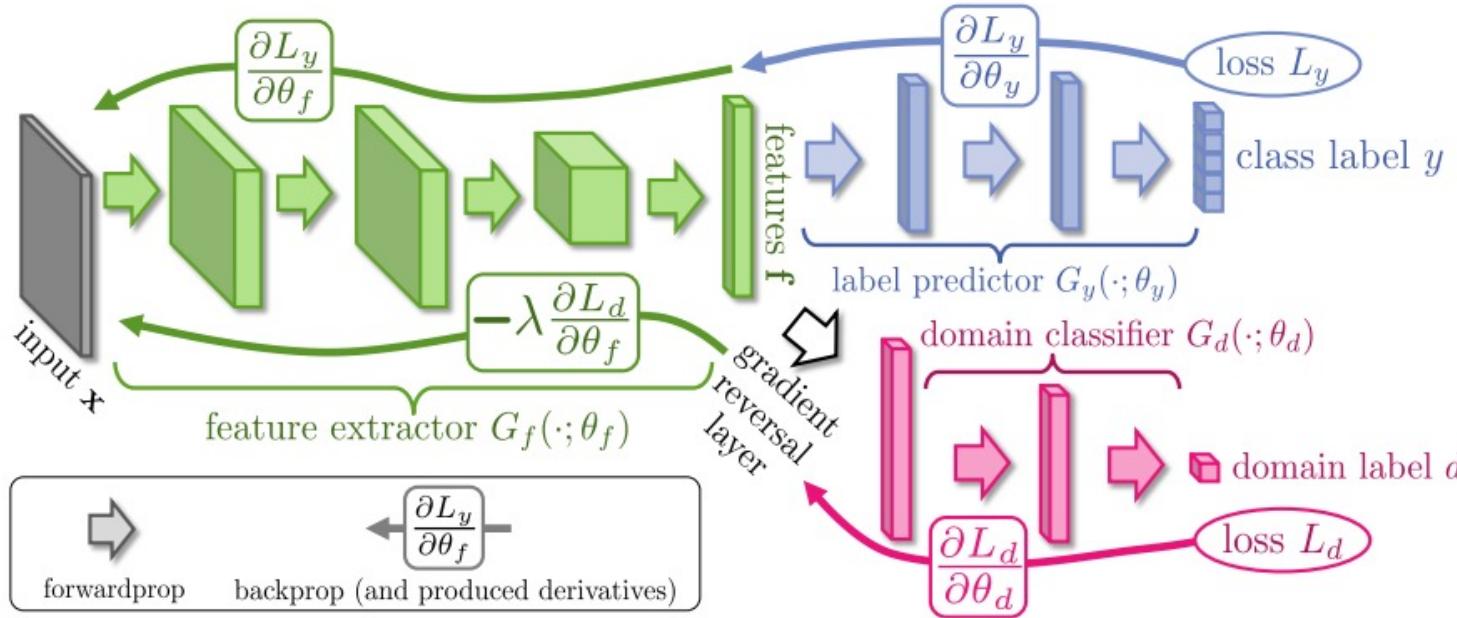
$$\epsilon_T(h) \leq \epsilon_S(h) + \frac{1}{2} d_{H\Delta H} \left(Q_X^{(m)}, P_X^{(m)} \right) + \epsilon_{joint}$$

(Target error \leq source error + divergence + joint error)

The main result resulted in many practical methods (approximation methods) in order to use the concept of divergence in the training itself.

- Classical domain adaptation methods
 - Metric learning
 - Sample re-weighting
 - Subspace alignment
 - ...
- Deep Learning-based methods
 - **Nowadays an hot topic of research**

Domain adaptation: Ganin & Lempitsky method



- In general we want to learn a mapping (input embedding) such that performance on the task are maximised, but penalises the domain classification.

Image from: Ganin & Lempitsky, "Unsupervised Domain Adaptation by Backpropagation"



Domain adaptation
Domain generalization (OOD generalization)



Domain generalization (also called, Out-of-distribution (OOD) generalization)

The problem of domain generalization (also called, out-of-distribution (OOD) generalization) can be formalized as follows:

- Training: $K = |E|$ training domains
 - $P^{(e)} \sim \{(x_i^e, y_i^e)\}_{i=1}^{m_e}$
 - $1 \leq e \leq |E|$
- Goal: find $h \in H$ that performs well in an unseen domain $|E| + 1$
 - $P^{(K+1)} \sim \left\{ \left(x_i^{(K+1)}, y_i^{(K+1)} \right) \right\}_{i=1}^{m_{(K+1)}}$
- Minimize the risk in the new environment
 - $R^{(K+1)}(h) = \mathbb{E}_{(x,y) \sim P^{(K+1)}} [L(h(x), y)]$

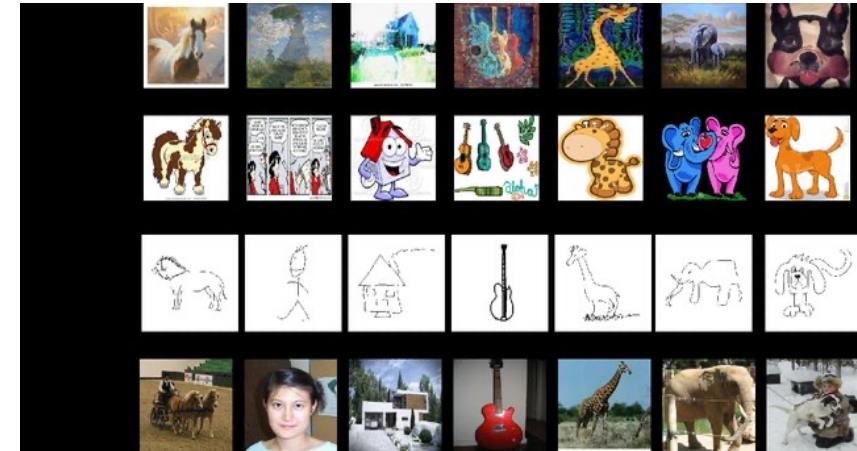
Note: also in this setup, different environments need to be “related” to each other.

DomainNet



- <http://ai.bu.edu/M3SDA/>
- 345 classes
- Domains: clipart, real, sketch, infograph, paintings, drawings

PACS



- <https://paperswithcode.com/dataset/pacs>
- 7 categories
- Domains: photo, paintings, cartoon, sketch

Method 1: Baseline method.

We call “Baseline” method the approach that consists simply on minimizing the error on the available domains.

- **Training:** $\min_f \frac{1}{K} \sum_{j=1}^k \mathbb{E}_{(x,y) \sim P^{(j)}} [L(f(x), y)]$
- **Test:** $\mathbb{E}_{(x,y) \sim P^{(K+1)}} [L(f(x), y)]$
- “Do nothing” method

Method 2: Invariant representation.

Learn a representation that is invariant across different domains

- Use domain adversarial neural networks (DANN)
 - ϕ (feature extraction)
 - $\omega \circ \phi$ (label classification)
 - $c \circ \phi$ (domain classification)
 - $loss = \frac{1}{K} \sum_{j=1}^K L(\omega \circ \phi(x), y) - \lambda \frac{1}{K} \sum_{j=1}^K L(c \circ \phi(x), y)$
 - $\min_{\phi, \omega} loss \quad \& \quad \max_c loss$
- “Do something” method



Lecture title

Recap



- **Domain adaptation**
 - **Unsupervised domain adaptation**
 - Main result
 - Practical methods
 - **Semi-supervised domain adaptation**
 - **Domain generalization**
 - Baseline method
 - Invariant representations method

