



Machine Learning Elements

An (In)formal Introduction

Riccardo Finotello

Artificial Intelligence and Modern Physics
A two-way connection

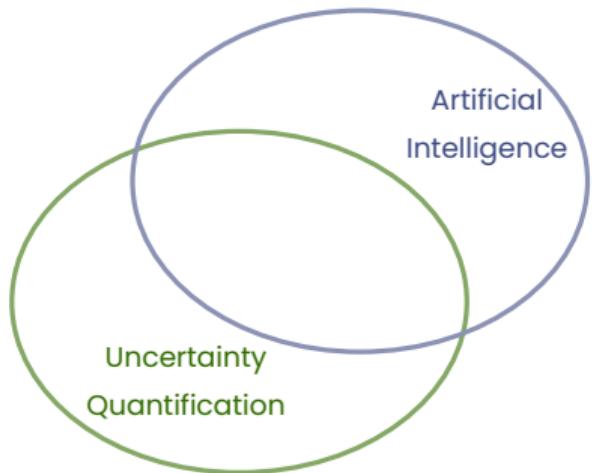
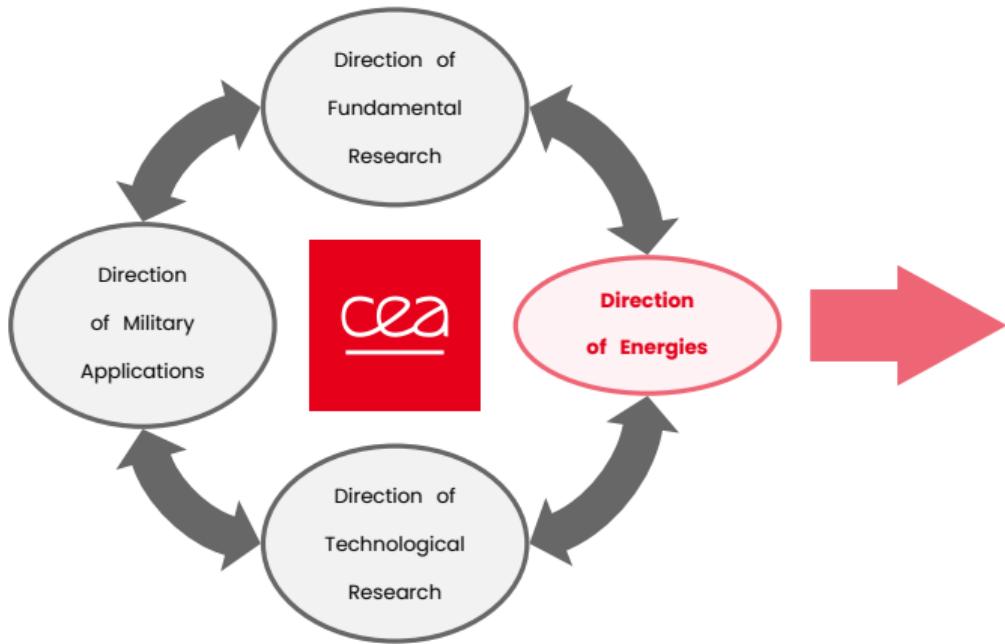
Monopoli (Bari, IT) | 29 September – 5 October 2024





Foreword

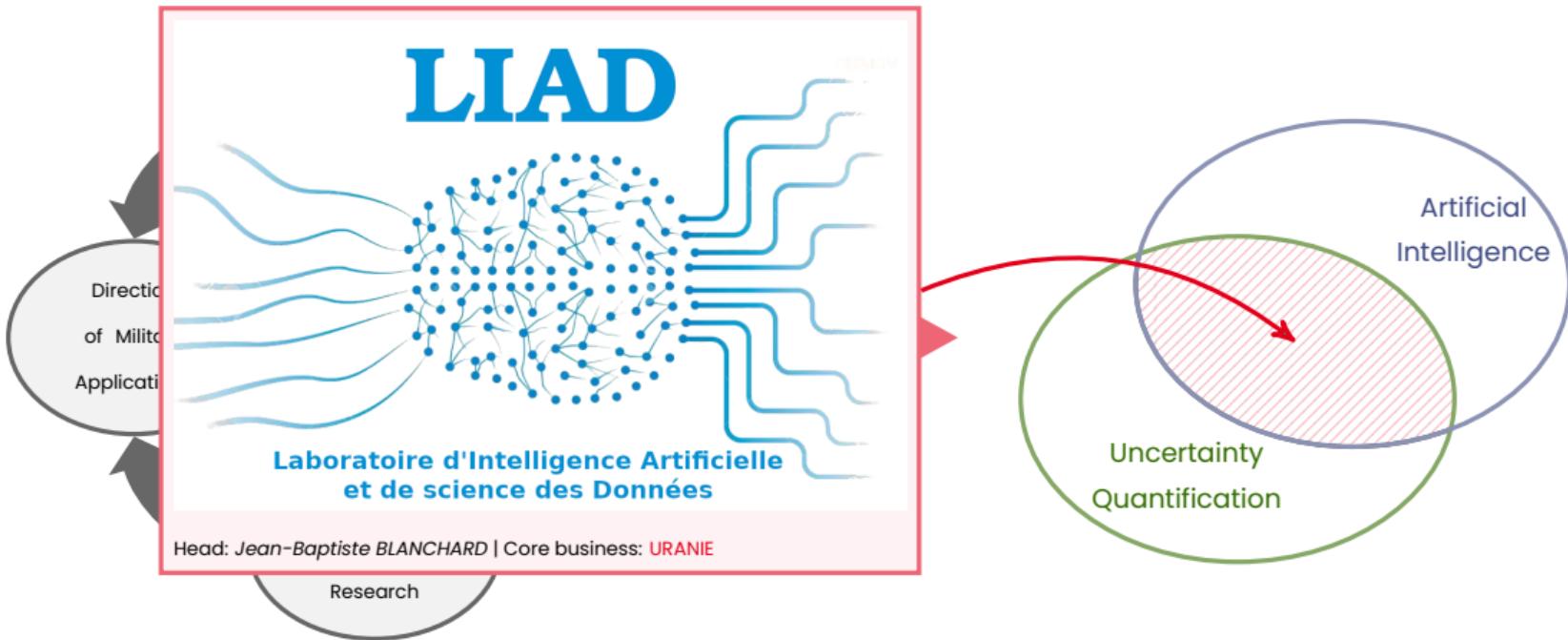
The LIAD Task Force





Foreword

The LIAD Task Force





Foreword

For the Hands-On Sessions

If you use git:

```
git clone https://github.com/thesfinox/aiphy-intro-ml-homework
```

Otherwise, download the latest **release** and unzip it.



Foreword

For the Hands-On Sessions

If you use git:

```
git clone https://github.com/thesfinox/aiphy-intro-ml-homework
```

Otherwise, download the latest [release](#) and unzip it.

Then proceed to the directory of the session (by default `aiphy-intro-ml-homework`) and install the requirements:

1. change path → `cd /path/to/the/directory`
2. create virtual environment → `python -m venv .venv`
3. activate the environment → `source .venv/bin/activate`
4. ensure package manager is installed → `python -m ensurepip --upgrade`
5. install the dependencies → `pip install -r requirements.txt`
6. check installation → `python -c "import numpy as np; print(np.__version__)"`

Python version at the time of writing: 3.10.12 (though any recent version should be fine...)

Should you have any issue, do not hesitate to ask. Remember to install [Python](#) and add it to your PATH if you work under Windows.

Table of contents

1. Some History and Philosophy to Start

What is ML? Some philosophical considerations...

2. The ML Mindset

- Data preparation
- Validation and test sets
- The variance vs bias trade-off
- Learning objectives and loss functions
- Regularisation

3. ML Algorithms

- Taxonomy of algorithms
- Unsupervised learning
- Supervised learning
- Ensemble learning

4. Neural Networks

- Computational graphs
- Non Linearity of Neural Networks
- Approximation theorems
- Neural network training
- Regularisation of neural networks

5. Conclusions

- Concluding remarks





1. Some History and Philosophy to Start

What is ML? Some philosophical considerations...

Table of contents

1. Some History and Philosophy to Start

What is ML? Some philosophical considerations...

2. The ML Mindset

3. ML Algorithms

4. Neural Networks

5. Conclusions



“

Programming computers to learn from
experience should eventually eliminate
the need for much of this detailed
programming effort ”

Arthur Samuel, IBM (1959)



What is ML?

The historical concept

Birth of the term

A. L. Samuel

Some Studies in Machine Learning Using the Game of Checkers

Abstract: Two machine-learning procedures have been investigated in some detail using the game of checkers. Enough work has been done to verify the fact that a computer can be programmed so that it will learn to play a better game of checkers than can be played by the person who wrote the program. Furthermore, it can learn to do this in a remarkably short period of time (8 or 10 hours of machine-playing time) when given only the rules of the game, a sense of direction, and a redundant and incomplete list of parameters which are thought to have something to do with the game, but whose correct signs and relative weights are unknown and unspecified. The principles of machine learning verified by these experiments are, of course, applicable to many other situations.

Arthur Samuel, IBM Journal (1959)





What is ML?

The historical concept

Birth of the term

A. L. Samuel

Some Studies in Machine Learning Using the Game of Checkers

Abstract: Two machine-learning procedures have been investigated in some detail using the game of checkers. Enough work has been done to verify the fact that a computer can be programmed so that it will learn to play a better game of checkers than can be played by the person who wrote the program. Furthermore, it can learn to do this in a remarkably short period of time (8 or 10 hours of machine-playing time) when given only the rules of the game, a sense of direction, and a redundant and incomplete list of parameters which are thought to have something to do with the game, but whose correct signs and relative weights are unknown and unspecified. The principles of machine learning verified by these experiments are, of course, applicable to many other situations.

Arthur Samuel, IBM Journal (1959)

1. DATA

2. LEARN

3. PREDICT



What is ML?

The historical concept

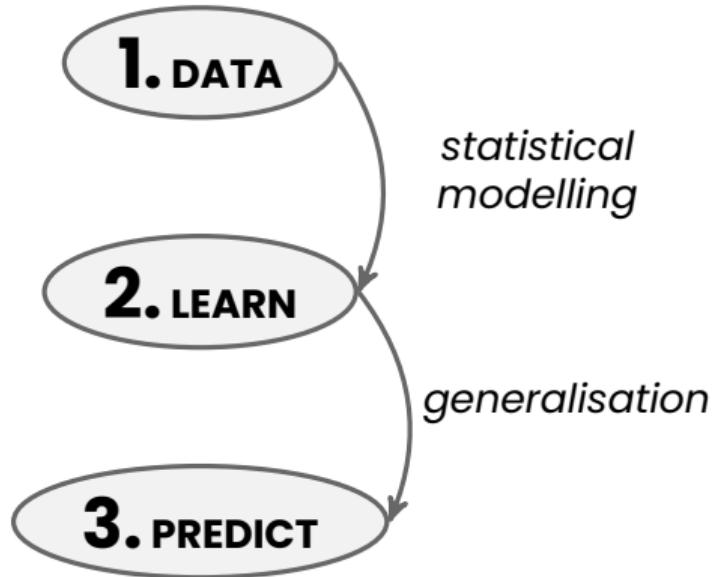
Birth of the term

A. L. Samuel

Some Studies in Machine Learning Using the Game of Checkers

Abstract: Two machine-learning procedures have been investigated in some detail using the game of checkers. Enough work has been done to verify the fact that a computer can be programmed so that it will learn to play a better game of checkers than can be played by the person who wrote the program. Furthermore, it can learn to do this in a remarkably short period of time (8 or 10 hours of machine-playing time) when given only the rules of the game, a sense of direction, and a redundant and incomplete list of parameters which are thought to have something to do with the game, but whose correct signs and relative weights are unknown and unspecified. The principles of machine learning verified by these experiments are, of course, applicable to many other situations.

Arthur Samuel, IBM Journal (1959)





What is ML?

Intuition and nested definitions

AI

■ Artificial Intelligence

- “human behaviour” emulation
- pattern recognition
- learning processes
- decision making



What is ML?

Intuition and nested definitions



■ Artificial Intelligence

- “human behaviour” emulation
- pattern recognition
- learning processes
- decision making

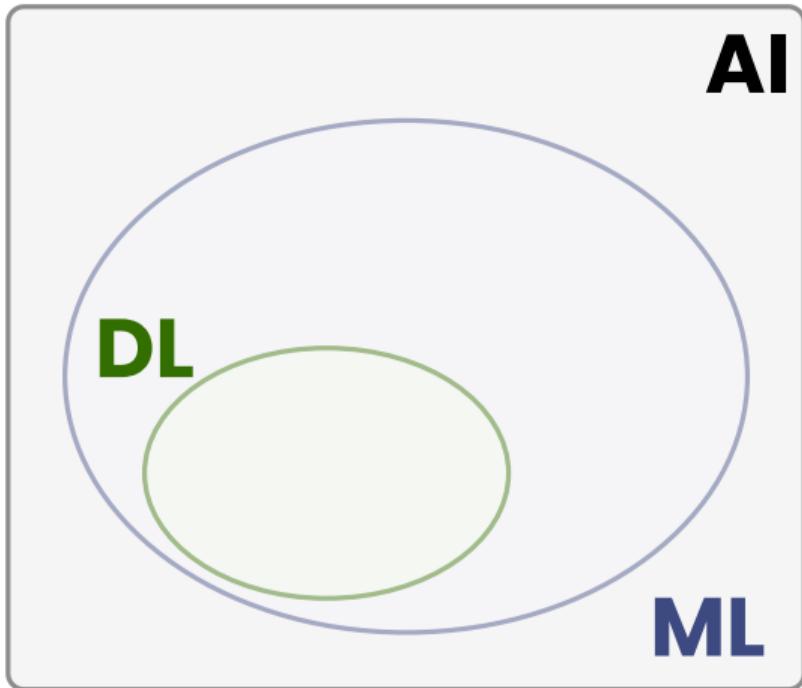
■ Machine Learning

- data exploitation
- statistical modelling
- generalisation on new data



What is ML?

Intuition and nested definitions



■ Artificial Intelligence

- “human behaviour” emulation
- pattern recognition
- learning processes
- decision making

■ Machine Learning

- data exploitation
- statistical modelling
- generalisation on new data

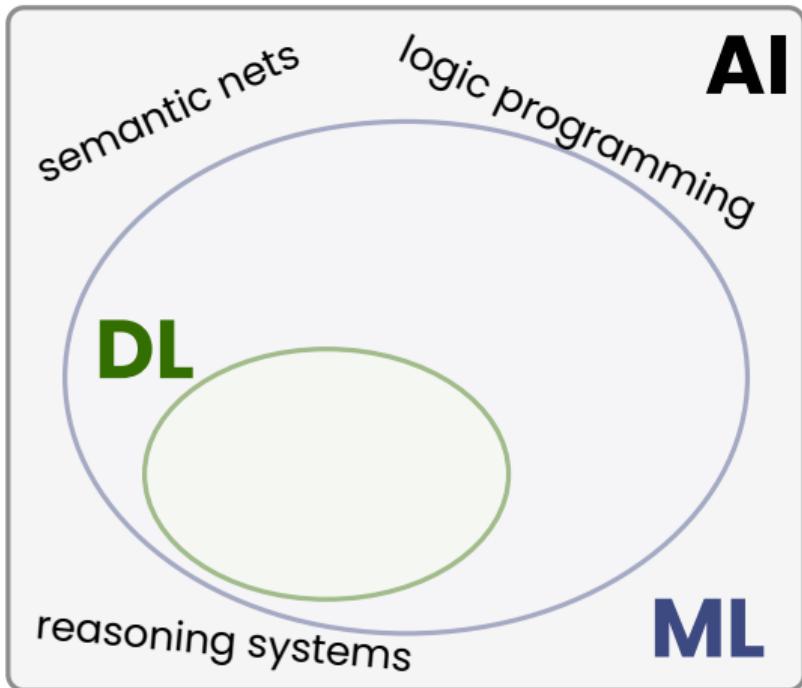
■ Deep Learning

- complex pattern processing
- highly non linear problems
- “mimic” human brains



What is ML?

Intuition and nested definitions



■ Artificial Intelligence

- “human behaviour” emulation
- pattern recognition
- learning processes
- decision making

■ Machine Learning

- data exploitation
- statistical modelling
- generalisation on new data

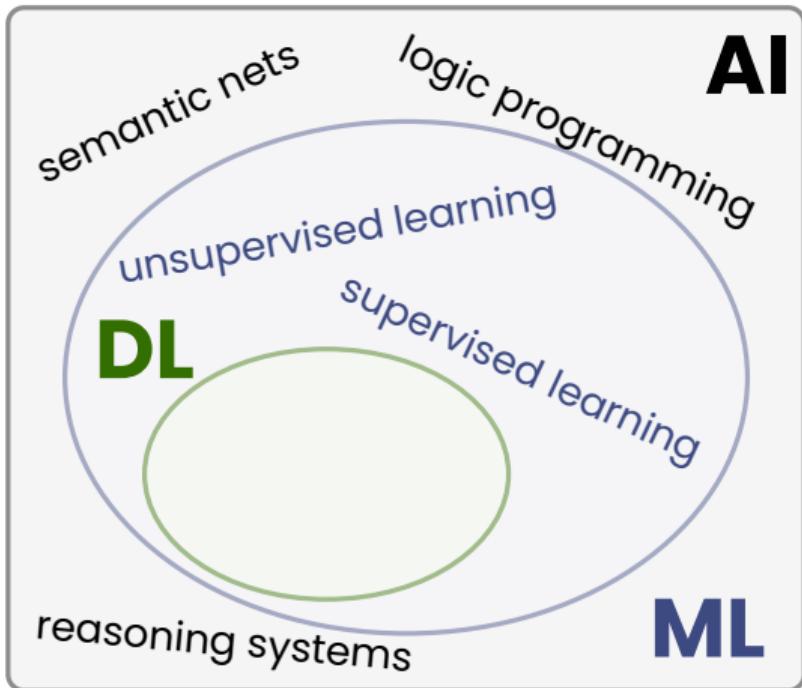
■ Deep Learning

- complex pattern processing
- highly non linear problems
- “mimic” human brains



What is ML?

Intuition and nested definitions



■ Artificial Intelligence

- “human behaviour” emulation
- pattern recognition
- learning processes
- decision making

■ Machine Learning

- data exploitation
- statistical modelling
- generalisation on new data

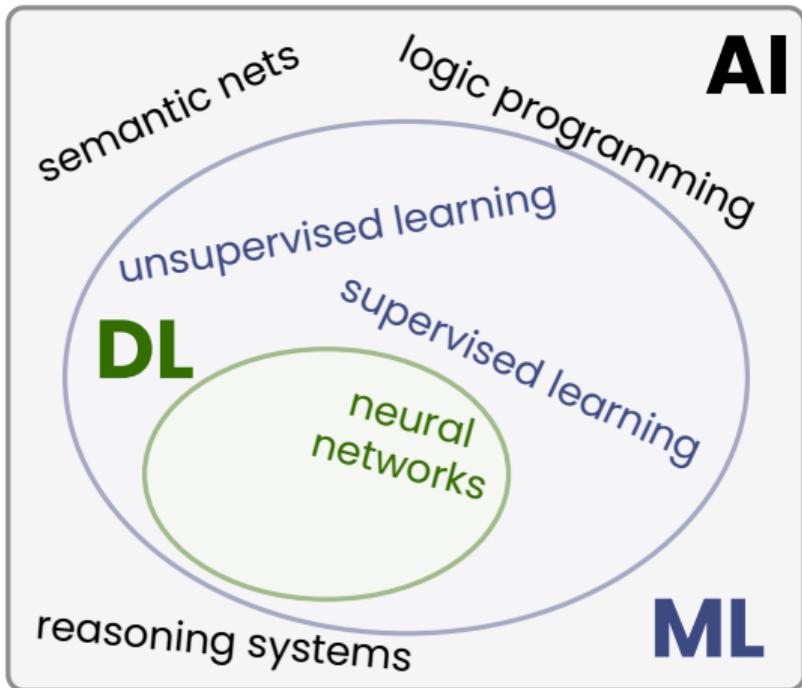
■ Deep Learning

- complex pattern processing
- highly non linear problems
- “mimic” human brains



What is ML?

Intuition and nested definitions



■ Artificial Intelligence

- "human behaviour" emulation
- pattern recognition
- learning processes
- decision making

■ Machine Learning

- data exploitation
- statistical modelling
- generalisation on new data

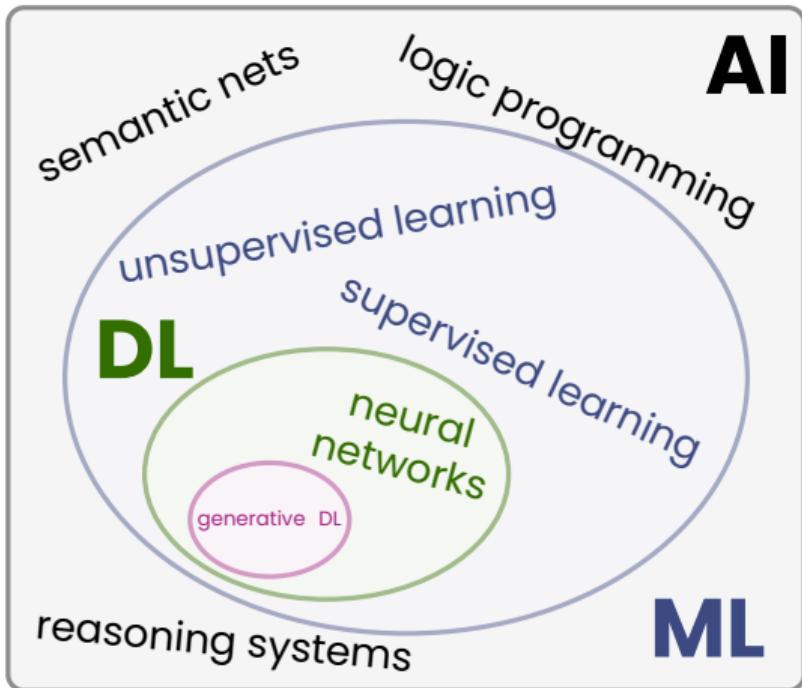
■ Deep Learning

- complex pattern processing
- highly non linear problems
- "mimic" human brains



What is ML?

Intuition and nested definitions



■ Artificial Intelligence

- “human behaviour” emulation
- pattern recognition
- learning processes
- decision making

■ Machine Learning

- data exploitation
- statistical modelling
- generalisation on new data

■ Deep Learning

- complex pattern processing
- highly non linear problems
- “mimic” human brains

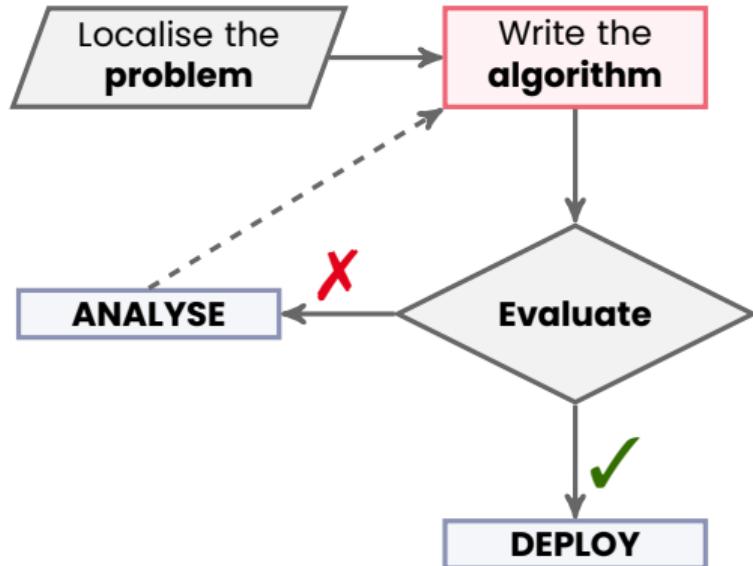


What is ML?

Intuitive behaviour

Write a **spam** filter:

- you know what characterises an email as spam
- you write a set of rules to flag emails as spam
- you evaluate your algorithm and decide whether to deploy it or reassess



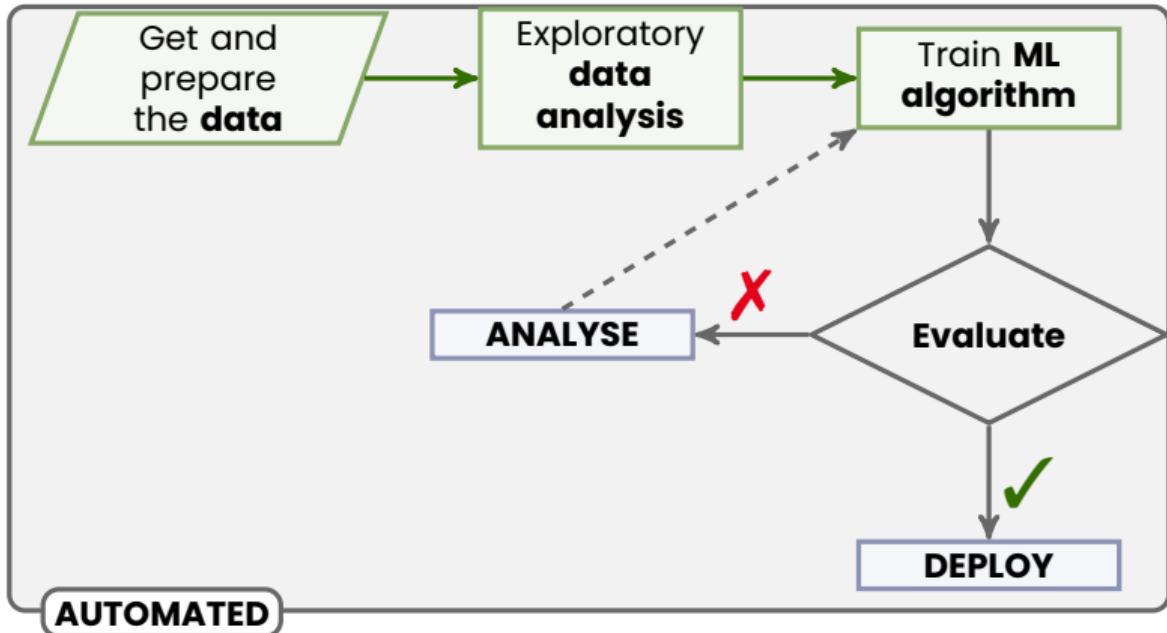


What is ML?

Intuitive behaviour

Write a **spam** filter:

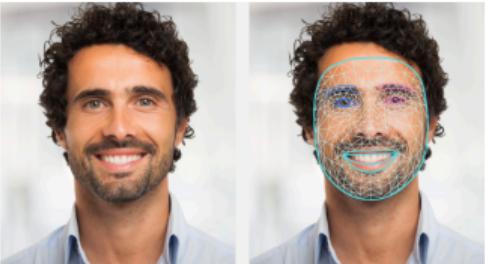
- you prepare a data tidying **pipeline**
- you write a ML training **pipeline** to flag emails as spam
- you write an evaluation **pipeline** and decide whether to deploy the model or reassess



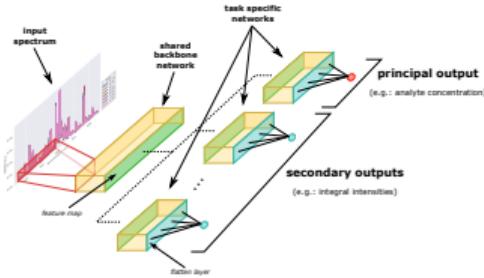


Why ML?

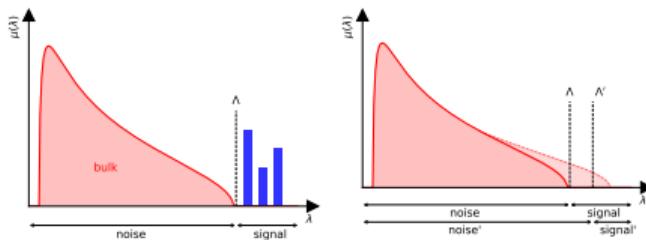
Real world scenarios



Mediapipe face landmark by



Trustworthy laser-induced breakdown spectroscopy ([arXiv:2210.03762](https://arxiv.org/abs/2210.03762))



Detectron2 panoptic segmentation / 3D pose estimation model by

Signal detection via *functional renormalization group* ([arXiv:2310.07499](https://arxiv.org/abs/2310.07499))



2. The ML Mindset



Data preparation

Table of contents

1. Some History and Philosophy to Start

2. The ML Mindset

Data preparation

Validation and test sets

The variance vs bias trade-off

Learning objectives and loss functions

Regularisation

3. ML Algorithms

4. Neural Networks

5. Conclusions





“

Paradoxically, data is the most under-valued and de-glamorised aspect of AI ”

Sambasivan *et al.* CHI'21



The ML Mindset

Worst case scenario

“BAD” DATA

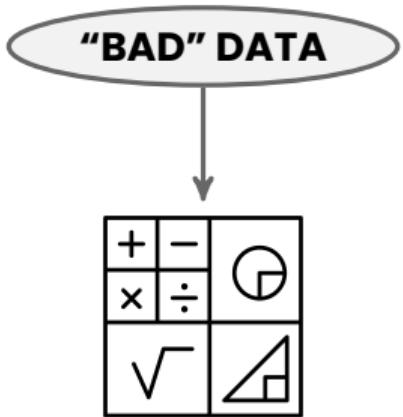
■ “Bad” input data

- *insufficient data* not enough data to learn anything useful
- *untidy data* bad missing data fillers, wrong categorical encoding, non representative samples, etc...
- *data leakage* the model “sees” the generalisation data
- *unbalanced dataset* the model develops a “bias”



The ML Mindset

Worst case scenario



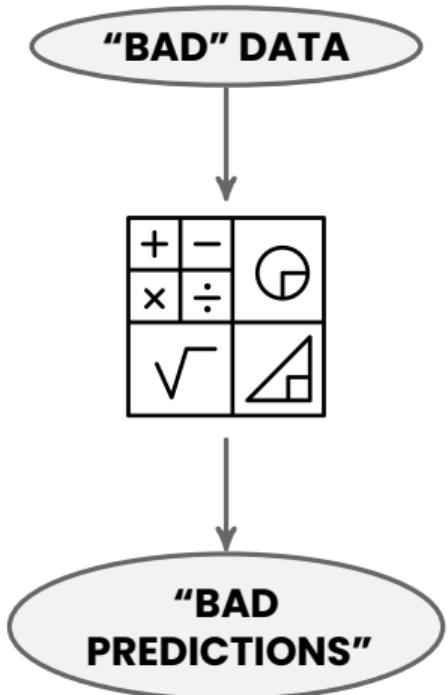
■ “Bad” input data

- ***insufficient data*** not enough data to learn anything useful
- ***untidy data*** bad missing data fillers, wrong categorical encoding, non representative samples, etc...
- ***data leakage*** the model “sees” the generalisation data
- ***unbalanced dataset*** the model develops a “bias”



The ML Mindset

Worst case scenario



■ “Bad” input data

- *insufficient data* not enough data to learn anything useful
- *untidy data* bad missing data fillers, wrong categorical encoding, non representative samples, etc...
- *data leakage* the model “sees” the generalisation data
- *unbalanced dataset* the model develops a “bias”

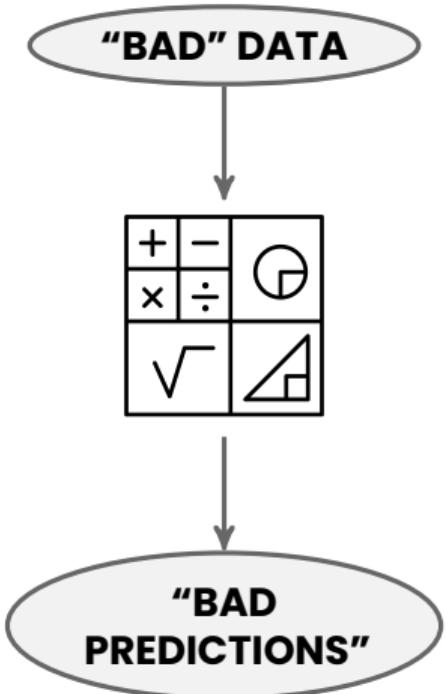
■ “Bad” predictions

- *biased model* cannot model input data \Rightarrow bad generalisation performance
- *overfitting* model is too “adapted” \Rightarrow bad generalisation performance



The ML Mindset

Worst case scenario



■ “Bad” input data

- *insufficient data* not enough data to learn anything useful
- *untidy data* bad missing data fillers, wrong categorical encoding, non representative samples, etc...
- *data leakage* the model “sees” the generalisation data
- *unbalanced dataset* the model develops a “bias”

■ “Bad” predictions

- *biased model* cannot model input data \Rightarrow bad generalisation performance
- *overfitting* model is too “adapted” \Rightarrow bad generalisation performance

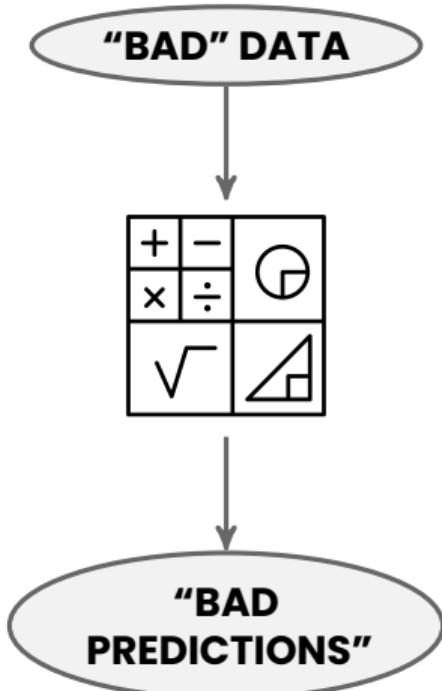
Garbage IN \Rightarrow Garbage OUT

Racial, sexist, and religious biases are **always** the result of human errors (even accidental)!

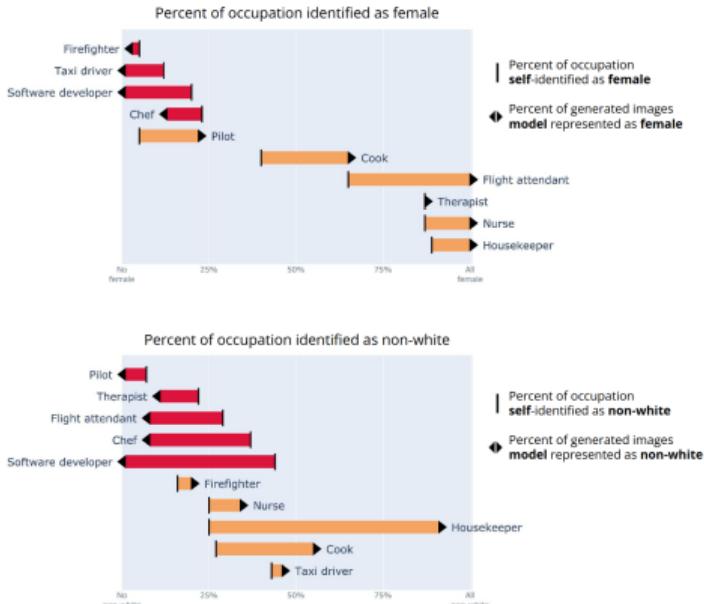


The ML Mindset

Worst case scenario



Simple examples of biases in AI



Bianchi et al. FAccT'23





The ML Mindset

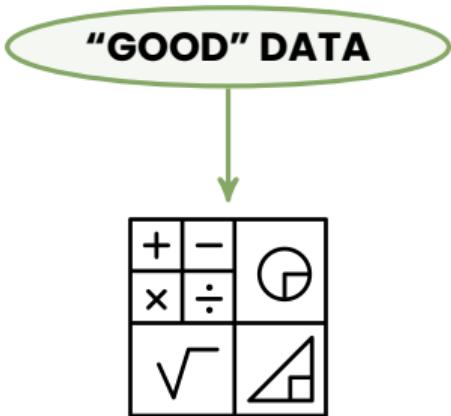
ML pipelines and working operations

"GOOD" DATA



The ML Mindset

ML pipelines and working operations



ML pipeline

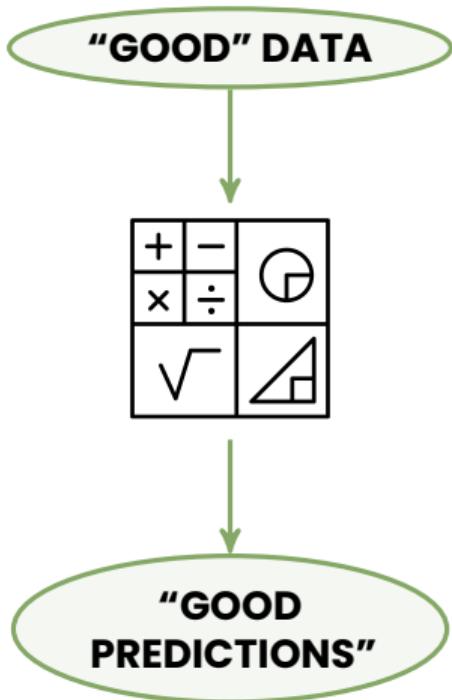
Introduce a set of "checklists" to...

- ...ensure high **data quality** (and tidiness);
- ...streamline **analysis** and **model building**;
- ...simplify the **learning** process and its **evaluation**;
- ...grant **reproducibility** and **experimentation**



The ML Mindset

ML pipelines and working operations



ML pipeline

Introduce a set of “checklists” to...

- ...ensure high **data quality** (and tidiness);
- ...streamline **analysis** and **model building**;
- ...simplify the **learning** process and its **evaluation**;
- ...grant **reproducibility** and **experimentation**

=

“ML PIPELINE”



The ML Mindset

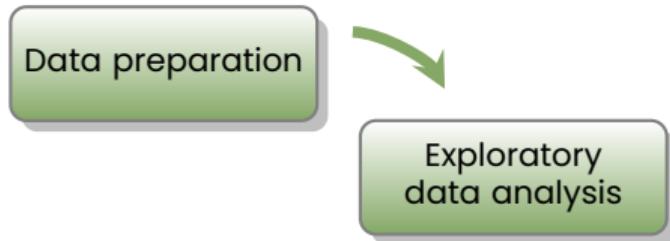
A clockwork pipeline

Data preparation



The ML Mindset

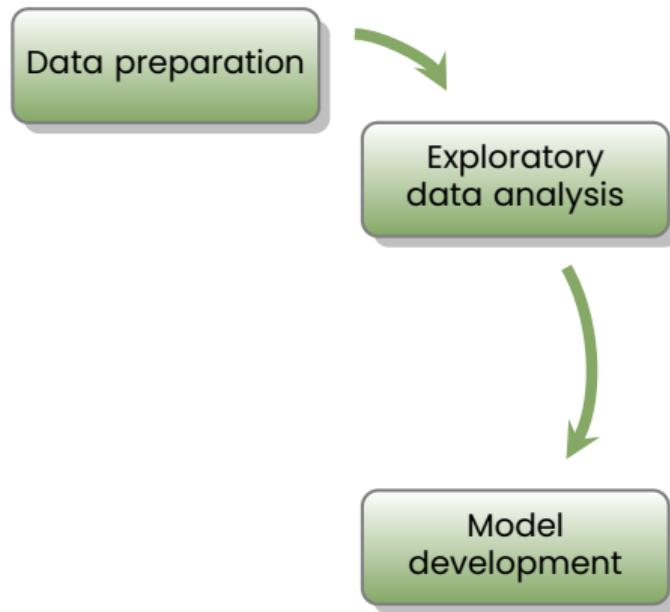
A clockwork pipeline





The ML Mindset

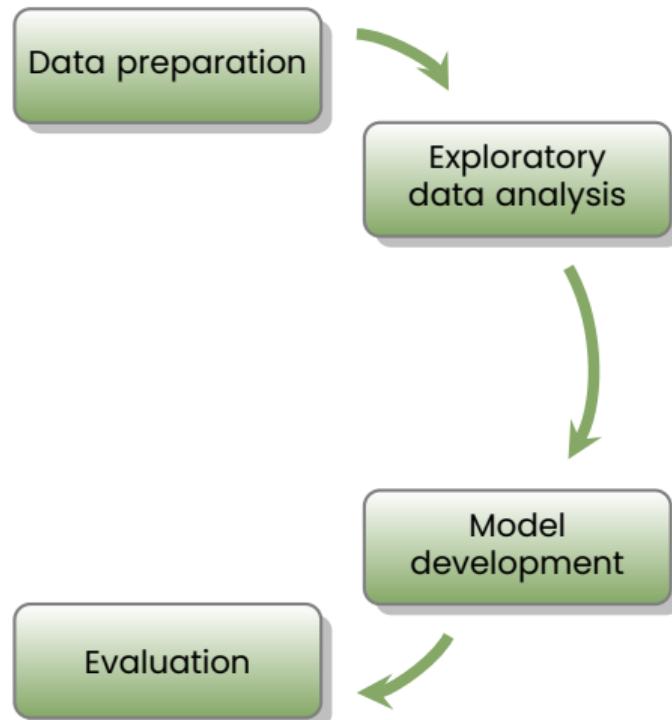
A clockwork pipeline





The ML Mindset

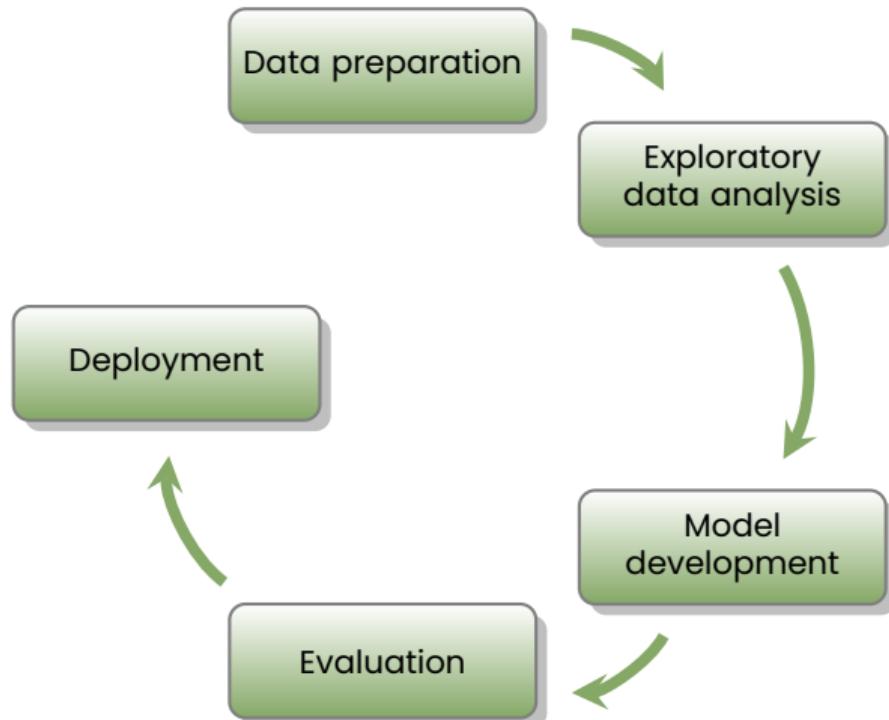
A clockwork pipeline





The ML Mindset

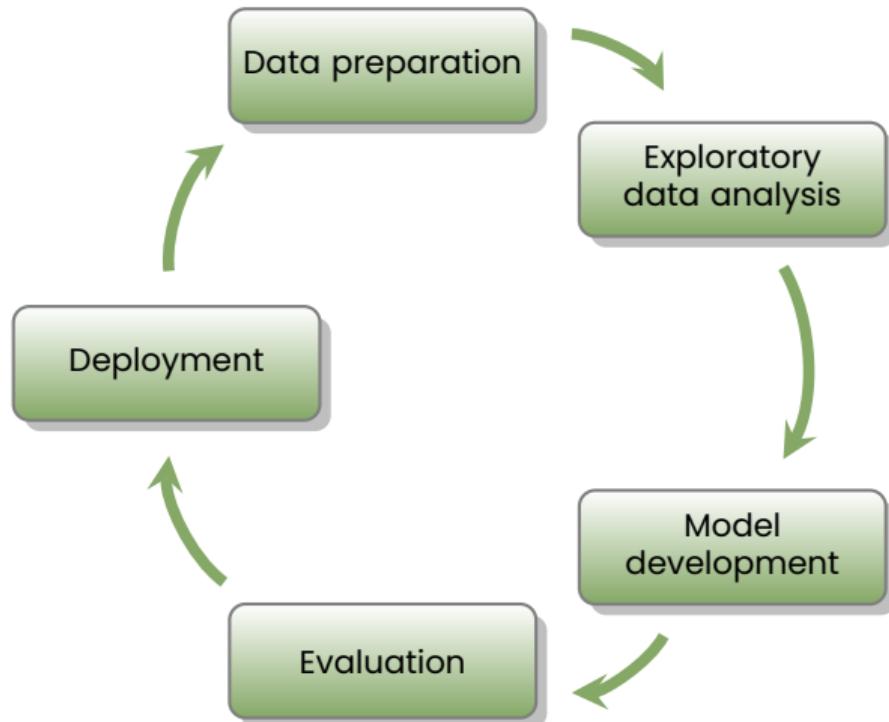
A clockwork pipeline





The ML Mindset

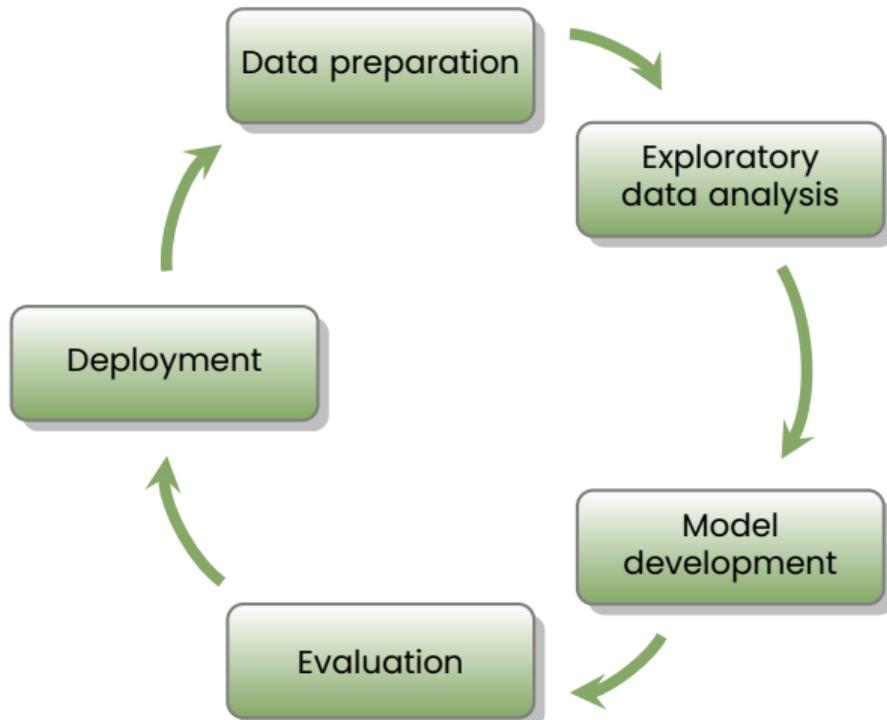
A clockwork pipeline





The ML Mindset

A clockwork pipeline



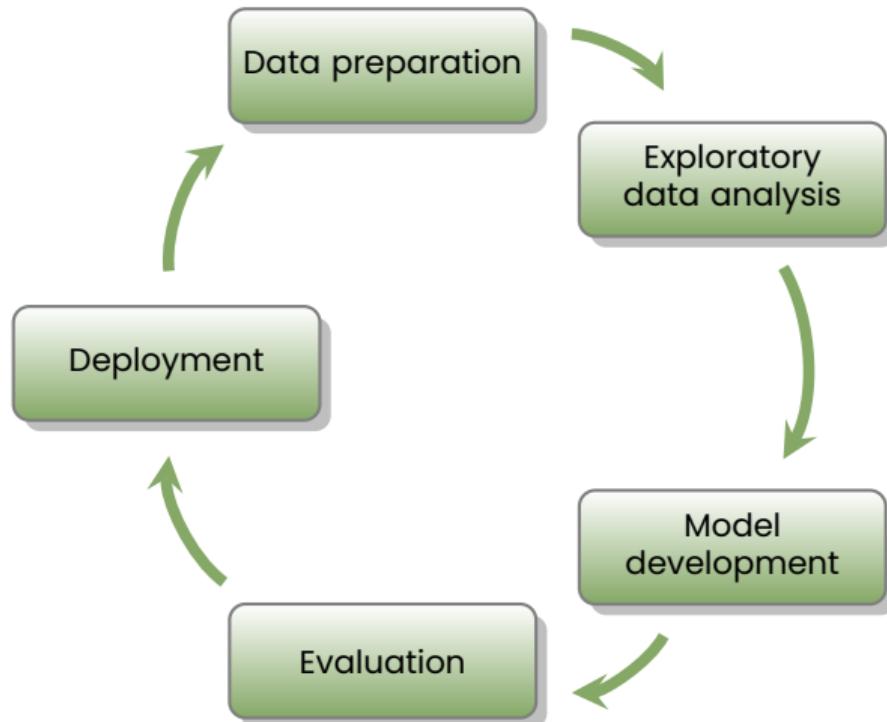
(xkcd.com)





The ML Mindset

A clockwork pipeline

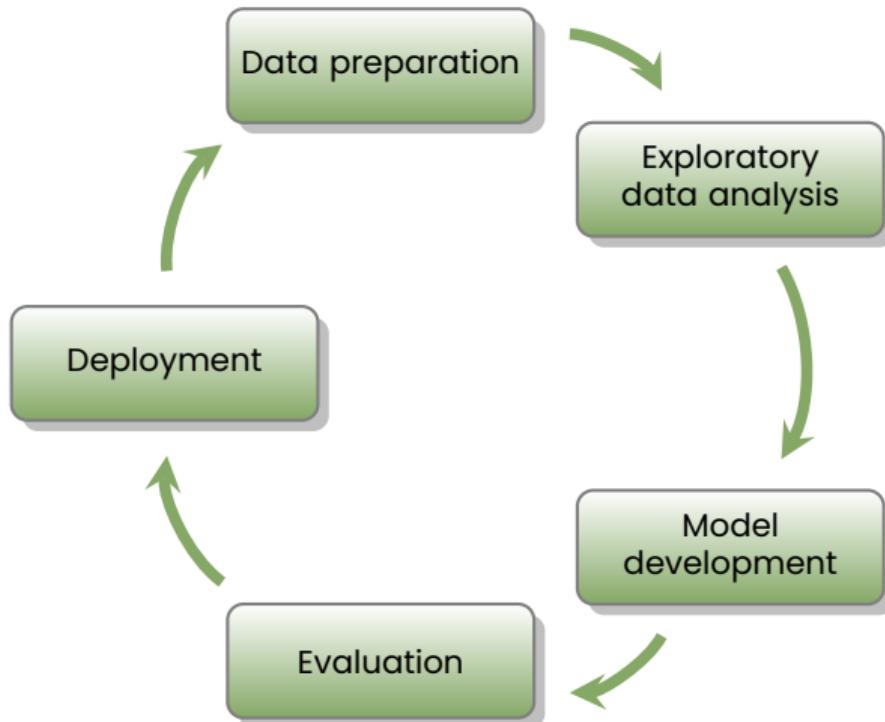


■ **modular** break down complex problems into
small bricks



The ML Mindset

A clockwork pipeline

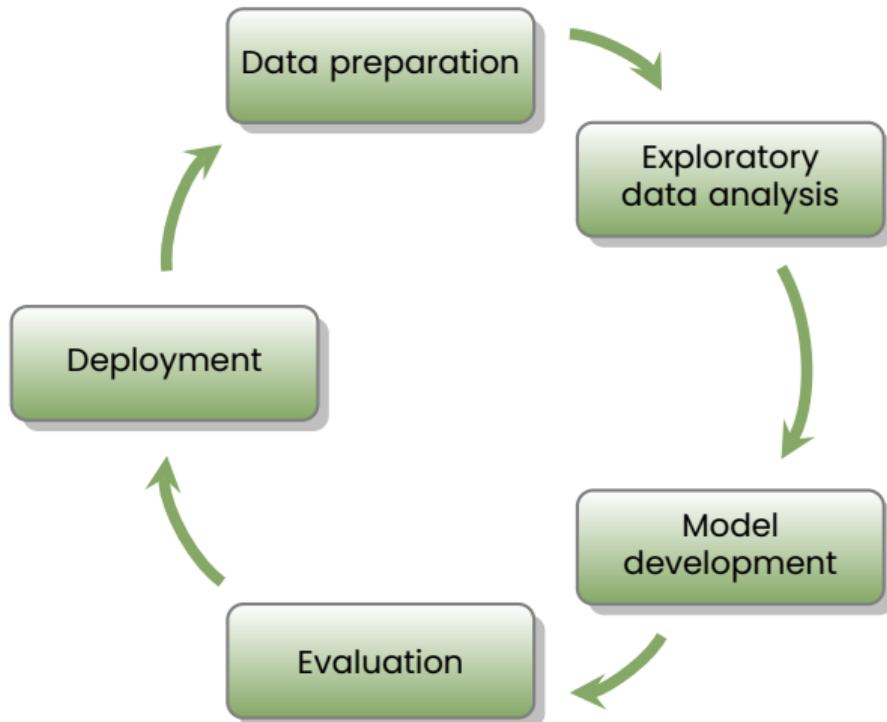


- **modular** break down complex problems into small bricks
- **reproducible** trace back analysis to well-defined breakpoints



The ML Mindset

A clockwork pipeline

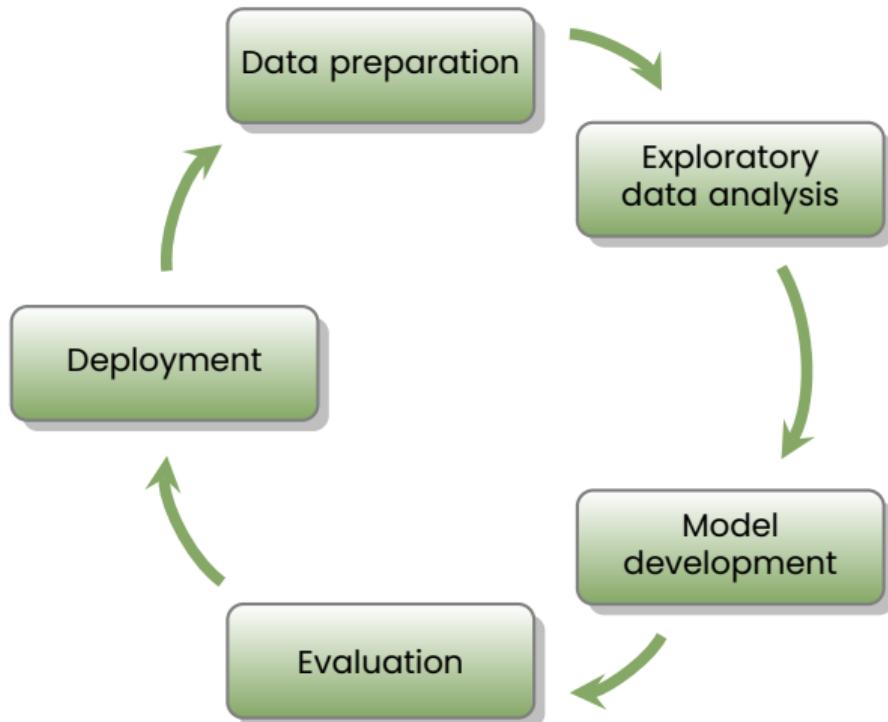


- **modular** break down complex problems into small bricks
- **reproducible** trace back analysis to well-defined breakpoints
- **experimental** trial-and-error is permitted and easier to implement



The ML Mindset

A clockwork pipeline



- **modular** break down complex problems into small bricks
- **reproducible** trace back analysis to well-defined breakpoints
- **experimental** trial-and-error is permitted and easier to implement
- **collaborative** agreeing on sensible choices enables peaceful and fruitful collaborations

The *Scikit-learn* guideline

“Improving the documentation is no less important than improving the library itself.”



2. The ML Mindset

- Validation and test sets

Table of contents

1. Some History and Philosophy to Start

2. The ML Mindset

Data preparation

Validation and test sets

The variance vs bias trade-off

Learning objectives and loss functions

Regularisation

3. ML Algorithms

4. Neural Networks

5. Conclusions





Dealing with Data

A generic beginning of a project

Let $\mathcal{D}_N = \{(\vec{x}_i, \vec{y}_i) \mid \mathbb{K}^p \ni \vec{x}_i \sim \mathcal{P}(X), \mathbb{K}^q \ni \vec{y}_i \sim \mathcal{P}(Y) \quad \forall i = 1, 2, \dots, N\}$:

\mathcal{D}_N

Some definitions to start:

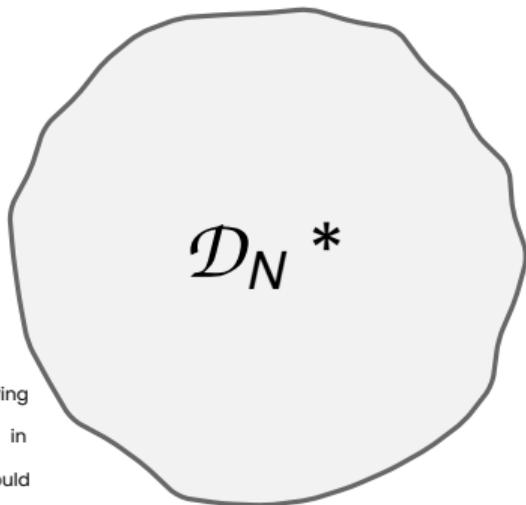
- \vec{x} are called **features** / exogenous variables / regressors / predictors / explanatory variables
- \vec{y} are called **labels** / endogenous variables / regressands / targets / explained variables



Dealing with Data

A generic beginning of a project

Let $\mathcal{D}_N = \{(\vec{x}_i, \vec{y}_i) \mid \mathbb{K}^p \ni \vec{x}_i \sim \mathcal{P}(X), \mathbb{K}^q \ni \vec{y}_i \sim \mathcal{P}(Y) \quad \forall i = 1, 2, \dots, N\}$:



* Imagine if an ordering
"by label" y_i was left in
the dataset: what would
happen in the following?

- **shuffle the dataset to avoid biases:**
any possible ordering of the data
should **never interfere**



Dealing with Data

A generic beginning of a project

Let $\mathcal{D}_N = \{(\vec{x}_i, \vec{y}_i) \mid \mathbb{K}^p \ni \vec{x}_i \sim \mathcal{P}(X), \mathbb{K}^q \ni \vec{y}_i \sim \mathcal{P}(Y) \quad \forall i = 1, 2, \dots, N\}$:

\mathcal{D}_N

- **shuffle the dataset to avoid biases:** any possible ordering of the data should **never interfere**



Dealing with Data

A generic beginning of a project

Let $\mathcal{D}_N = \{(\vec{x}_i, \vec{y}_i) \mid \mathbb{K}^p \ni \vec{x}_i \sim \mathcal{P}(X), \mathbb{K}^q \ni \vec{y}_i \sim \mathcal{P}(Y) \quad \forall i = 1, 2, \dots, N\}$:



The number of samples to leave in the splits is highly dependent on the size of the dataset, type of task, computing resources, regularity of the data, etc... Traditionally, smaller datasets may require $\sim 80\%$ of training data.

However, **Big Data** may take up to 99% of training data, as the test set will remain statistically relevant.

(e.g. see Andrew Ng (2019))

- **shuffle the dataset** to avoid **biases**: any possible ordering of the data should **never interfere**
- prepare a **test set** and “hide” it until your final evaluation

Data leakage

The **test set** should be **randomly** and **independently** chosen to represent “real-world” data. It must **never** come into contact with **training** procedures.



Dealing with Data

A generic beginning of a project

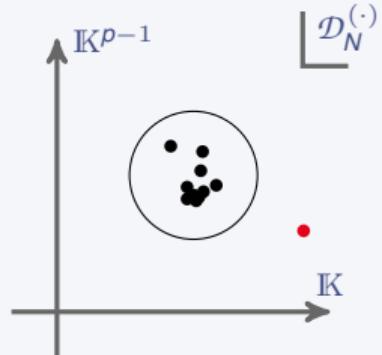
Let $\mathcal{D}_N = \{(\vec{x}_i, \vec{y}_i) \mid \mathbb{K}^p \ni \vec{x}_i \sim \mathcal{P}(X), \mathbb{K}^q \ni \vec{y}_i \sim \mathcal{P}(Y) \quad \forall i = 1, 2, \dots, N\}$:



The number of samples to leave in the splits is highly dependent on the size of the dataset, type of task, computing resources, regularity of the data, etc... Traditionally, smaller datasets may require $\sim 80\%$ of training data. However, **Big Data** may take up to 99% of training data, as the test set will remain statistically relevant.

(e.g. see [Andrew Ng \(2019\)](#))

Food for thought... Outliers?



1. you have some **pictures**, and outliers are overexposed samples you would get rid of anyway
2. you analyse **financial** data where stock returns are capped to a given value
3. you are building a **cybersecurity** defence and outliers are attacked data
4. you have **scientific** data, and you are trying to derive an analytical formula using insights from ML



Dealing with Data

A generic beginning of a project

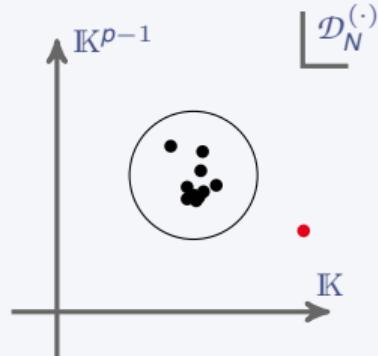
Let $\mathcal{D}_N = \{(\vec{x}_i, \vec{y}_i) \mid \mathbb{K}^p \ni \vec{x}_i \sim \mathcal{P}(X), \mathbb{K}^q \ni \vec{y}_i \sim \mathcal{P}(Y) \quad \forall i = 1, 2, \dots, N\}$:



The number of samples to leave in the splits is highly dependent on the size of the dataset, type of task, computing resources, regularity of the data, etc... Traditionally, smaller datasets may require $\sim 80\%$ of training data. However, **Big Data** may take up to 99% of training data, as the test set will remain statistically relevant.

(e.g. see [Andrew Ng \(2019\)](#))

Food for thought... Outliers?



1. you have some **pictures**, and outliers are overexposed samples you would get rid of anyway
2. you analyse **financial** data where stock returns are capped to a given value
3. you are building a **cybersecurity** defence and outliers are attacked data
4. you have **scientific** data, and you are trying to derive an analytical formula using insights from ML

1. remove them everywhere (they do not represent real-world data)
2. leave them everywhere / move them to $\mathcal{D}_N^{(\text{test})}$ and cap the value
3. move them to $\mathcal{D}_N^{(\text{test})}$ (learn difficult cases, to predict easy samples)
4. move them to $\mathcal{D}_N^{(\text{test})}$ (the model should be able to predict them anyway)



Dealing with Data

The need of self-evaluation

$$\mathcal{D}_N^{(\text{dev})}$$

Let us suppose:

- good **exploratory data analysis**
- model **dev. and training** on $\mathcal{D}_N^{(\text{dev})}$
- **no bias / sensible** choices
- **good** performance
- in general... **nothing strange**



Dealing with Data

The need of self-evaluation

$$\mathcal{D}_N^{(\text{dev})}$$
$$\mathcal{D}_N^{(\text{test})}$$

Let us suppose:

- good **exploratory data analysis**
- model **dev. and training** on $\mathcal{D}_N^{(\text{dev})}$
- **no bias / sensible** choices
- **good** performance
- in general... **nothing strange**

You get:

- **Bad generalisation** performance
- **Biased/unbalanced** results



*...lost in time,
like tears in rain*

Roy Batty / Blade Runner (1982)



Dealing with Data

The need of self-evaluation

$$\mathcal{D}_N^{(\text{dev})}$$
$$\mathcal{D}_N^{(\text{test})}$$

Let us suppose:

- good **exploratory data analysis**
- model **dev. and training** on $\mathcal{D}_N^{(\text{dev})}$
- **no bias / sensible** choices
- **good** performance
- in general... **nothing strange**

You get:

- **Bad generalisation** performance
- **Biased/unbalanced** results

Most probably, the model **overfits** (see later) $\mathcal{D}_N^{(\text{dev})}$ and is unable to **statistically** represent new data!



Dealing with Data

The need of self-evaluation

$$\mathcal{D}_N^{(\text{dev})}$$
$$\mathcal{D}_N^{(\text{test})}$$

Let us suppose:

- good **exploratory data analysis**
- model **dev. and training** on $\mathcal{D}_N^{(\text{dev})}$
- **no bias / sensible** choices
- **good** performance
- in general... **nothing strange**

You get:

- **Bad generalisation** performance
- **Biased/unbalanced** results

Most probably, the model **overfits** (see later) $\mathcal{D}_N^{(\text{dev})}$ and is unable to **statistically** represent new data!

In other words, we need to **evaluate** the model before deploying it, or it will be, in general, a catastrophe!



Validation

The importance of choosing a validation set

Holdout validation



Cross validation (K-fold)

- build $\mathcal{D}_N^{(\text{val})} \subset \mathcal{D}_N^{(\text{dev})}$ and $\mathcal{D}_N^{(\text{train})} = \mathcal{D}_N^{(\text{dev})} \setminus \mathcal{D}_N^{(\text{val})}$ **once**
- computing time-**friendly**
- good for (very) **large datasets**
- **easy** to implement, **easy** to use usually,
no boilerplate code in Pytorch Lightning, Keras, Hugging Face, etc...

- more **robust** estimator
- first **insight** into **uncertainties**
- **time consuming** for **large** datasets
- might need some **coding** yes, scikit-learn

has a good implementation! Do not worry!





Validation

Validation error and how to use it

Let $\text{dist}(y, \hat{y})$ be a metric **distance** between **ground truth** y and its **prediction** \hat{y} (e.g. mean squared error, cross entropy, etc.) [N.B.: what said for a scalar y can be said for \hat{y}] and compute **error** E on $\mathcal{D}^{(\text{val})}$.



Validation

Validation error and how to use it

Let $\text{dist}(y, \hat{y})$ be a metric **distance** between **ground truth** y and its **prediction** \hat{y} (e.g. mean squared error, cross entropy, etc.) [N.B.: what said for a scalar y can be said for \vec{y}] and compute **error** \mathcal{E} on $\mathcal{D}^{(\text{val})}$.

Holdout validation

$$\begin{aligned}\mathcal{E}_{\mathcal{D}_N^{(\text{val})}}(y, \hat{y}) &= \mathbb{E}_{\mathcal{D}_N^{(\text{val})}} [\text{dist}(y, \hat{y})] \\ &= \frac{1}{m} \sum_{p=0}^{m-1} \text{dist}(y_p, \hat{y}_p).\end{aligned}$$



Validation

Validation error and how to use it

Let $\text{dist}(y, \hat{y})$ be a metric **distance** between **ground truth** y and its **prediction** \hat{y} (e.g. mean squared error, cross entropy, etc.) [N.B.: what said for a scalar y can be said for \vec{y}] and compute **error** E on $\mathcal{D}^{(\text{val})}$.

Holdout validation

$$\begin{aligned} E_{\mathcal{D}_N^{(\text{val})}}(y, \hat{y}) &= \mathbb{E}_{\mathcal{D}_N^{(\text{val})}} [\text{dist}(y, \hat{y})] \\ &= \frac{1}{m} \sum_{p=0}^{m-1} \text{dist}(y_p, \hat{y}_p). \end{aligned}$$

Cross validation (K-fold)

$$\begin{aligned} E_{\mathcal{D}_N^{(\text{val})}}^{(K)}(y, \hat{y}) &= \mathbb{E}_{\text{K-folds}} \left[\mathbb{E}_{\mathcal{D}_N^{(\text{val})}} [\text{dist}(y, \hat{y})] \right] \\ &= \frac{1}{K} \sum_{i=0}^{K-1} \frac{1}{m_i} \sum_{p=0}^{m_i-1} \text{dist}(y_p^{[i]}, \hat{y}_p^{[i]}). \end{aligned}$$

Notation: let $n = |\mathcal{D}_N^{(\text{dev})}|$, then $\lfloor \frac{n}{K} \rfloor \leq m_i \leq \lfloor \frac{n}{K} \rfloor + 1$ is the size of the i -th validation fold $\Rightarrow n - \frac{n}{K} = \frac{K-1}{K}n$ is the size of the remaining set.

The particular case $K = n$ is called "leave-one-out" validation.



Validation

Validation error and how to use it

Let $\text{dist}(y, \hat{y})$ be a metric **distance** between **ground truth** y and its **prediction** \hat{y} (e.g. mean squared error, cross entropy, etc.) [N.B.: what said for a scalar y can be said for \vec{y}] and compute **error** \mathcal{E} on $\mathcal{D}^{(\text{val})}$.

Holdout validation

$$\begin{aligned}\mathcal{E}_{\mathcal{D}_N^{(\text{val})}}(y, \hat{y}) &= \mathbb{E}_{\mathcal{D}_N^{(\text{val})}} [\text{dist}(y, \hat{y})] \\ &= \frac{1}{m} \sum_{p=0}^{m-1} \text{dist}(y_p, \hat{y}_p).\end{aligned}$$

Cross validation (K-fold)

$$\begin{aligned}\mathcal{E}_{\mathcal{D}_N^{(\text{val})}}^{(K)}(y, \hat{y}) &= \mathbb{E}_{\text{K-folds}} \left[\mathbb{E}_{\mathcal{D}_N^{(\text{val})}} [\text{dist}(y, \hat{y})] \right] \\ &= \frac{1}{K} \sum_{i=0}^{K-1} \frac{1}{m_i} \sum_{p=0}^{m_i-1} \text{dist}(y_p^{[i]}, \hat{y}_p^{[i]}).\end{aligned}$$

$\mathcal{E}_{\mathcal{D}_N^{(\text{val})}}(y, \hat{y})$ can be used as **estimate** of the **prediction error**

Notation: let $n = |\mathcal{D}_N^{(\text{dev})}|$, then $\lfloor \frac{n}{K} \rfloor \leq m_i \leq \lfloor \frac{n}{K} \rfloor + 1$ is the size of the i -th validation fold $\Rightarrow n - \frac{n}{K} = \frac{K-1}{K}n$ is the size of the remaining set.

The particular case $K = n$ is called "leave-one-out" validation.



Validation

Validation error and how to use it

Let $\text{dist}(y, \hat{y})$ be a metric **distance** between **ground truth** y and its **prediction** \hat{y} (e.g. mean squared error, cross entropy, etc.) [N.B.: what said for a scalar y can be said for \vec{y}] and compute **error** \mathcal{E} on $\mathcal{D}^{(\text{val})}$.

Holdout validation

$$\begin{aligned}\mathcal{E}_{\mathcal{D}_N^{(\text{val})}}(y, \hat{y}) &= \mathbb{E}_{\mathcal{D}_N^{(\text{val})}} [\text{dist}(y, \hat{y})] \\ &= \frac{1}{m} \sum_{p=0}^{m-1} \text{dist}(y_p, \hat{y}_p).\end{aligned}$$

$\mathcal{E}_{\mathcal{D}_N^{(\text{val})}}(y, \hat{y})$ can be used as **estimate** of the **prediction error**

Cross validation (K-fold)

$$\begin{aligned}\mathcal{E}_{\mathcal{D}_N^{(\text{val})}}^{(K)}(y, \hat{y}) &= \mathbb{E}_{\text{K-folds}} \left[\mathbb{E}_{\mathcal{D}_N^{(\text{val})}} [\text{dist}(y, \hat{y})] \right] \\ &= \frac{1}{K} \sum_{i=0}^{K-1} \frac{1}{m_i} \sum_{p=0}^{m_i-1} \text{dist}(y_p^{[i]}, \hat{y}_p^{[i]}).\end{aligned}$$

More complicated than that...The choice depends on K . Moreover, the whole $\mathcal{D}_N^{(\text{dev})}$ is used!

Notation: let $n = |\mathcal{D}_N^{(\text{dev})}|$, then $\lfloor \frac{n}{K} \rfloor \leq m_i \leq \lfloor \frac{n}{K} \rfloor + 1$ is the size of the i -th validation fold $\Rightarrow n - \frac{n}{K} = \frac{K-1}{K}n$ is the size of the remaining set.

The particular case $K = n$ is called "leave-one-out" validation.



Validation

Validation error and how to use it

Let $\text{dist}(y, \hat{y})$ be a metric **distance** between **ground truth** y and its **prediction** \hat{y} (e.g. mean squared error, cross entropy, etc.) [N.B.: what said for a scalar y can be said for \vec{y}] and compute **error** E on $\mathcal{D}^{(\text{val})}$.

Holdout validation

$$\begin{aligned} E_{\mathcal{D}_N^{(\text{val})}}(y, \hat{y}) &= \mathbb{E}_{\mathcal{D}_N^{(\text{val})}} [\text{dist}(y, \hat{y})] \\ &= \frac{1}{m} \sum_{p=0}^{m-1} \text{dist}(y_p, \hat{y}_p). \end{aligned}$$

$E_{\mathcal{D}_N^{(\text{val})}}(y, \hat{y})$ can be used as **estimate** of the **prediction error**

Cross validation (K-fold)

$$\begin{aligned} E_{\mathcal{D}_N^{(\text{val})}}^{(K)}(y, \hat{y}) &= \mathbb{E}_{\text{K-folds}} \left[\mathbb{E}_{\mathcal{D}_N^{(\text{val})}} [\text{dist}(y, \hat{y})] \right] \\ &= \frac{1}{K} \sum_{i=0}^{K-1} \frac{1}{m_i} \sum_{p=0}^{m_i-1} \text{dist}(y_p^{[i]}, \hat{y}_p^{[i]}). \end{aligned}$$

More complicated than that...The choice depends on K . Moreover, the whole $\mathcal{D}_N^{(\text{dev})}$ is used!
We will come back to this later...

Notation: let $n = |\mathcal{D}_N^{(\text{dev})}|$, then $\lfloor \frac{n}{K} \rfloor \leq m_i \leq \lfloor \frac{n}{K} \rfloor + 1$ is the size of the i -th validation fold $\Rightarrow n - \frac{n}{K} = \frac{K-1}{K}n$ is the size of the remaining set.

The particular case $K = n$ is called "leave-one-out" validation.



Validation

Parameters vs hyperparameters

Let $M = \{f^{(n)} \mid n = 1, 2, \dots\}$ set of *models* (e.g. linear model, support vector machine, decision tree, neural network, etc.)



Validation

Parameters vs hyperparameters

Let $M = \{f^{(n)} \mid n = 1, 2, \dots\}$ set of *models* (e.g. linear model, support vector machine, decision tree, neural network, etc.)

Each $f^{(n)}$ has two sets of dependencies $\Rightarrow f^{(n)} = f^{(n)}(\Theta; \Omega)$:

- Θ is the set of **parameters** (i.e. the weights of the model $\rightarrow y = \vec{\beta} \cdot \vec{x}$)
- Ω is the set of **hyperparameters** (i.e. constraints of the model $\rightarrow y = \vec{\beta} \cdot \vec{x} + \lambda \vec{\beta} \cdot \vec{\beta}$)



Validation

Parameters vs hyperparameters

Let $M = \{f^{(n)} \mid n = 1, 2, \dots\}$ set of *models* (e.g. linear model, support vector machine, decision tree, neural network, etc.)

Each $f^{(n)}$ has two sets of dependencies $\Rightarrow f^{(n)} = f^{(n)}(\Theta; \Omega)$:

- Θ is the set of **parameters** (i.e. the weights of the model $\rightarrow y = \vec{\beta} \cdot \vec{x}$)
- Ω is the set of **hyperparameters** (i.e. constraints of the model $\rightarrow y = \vec{\beta} \cdot \vec{x} + \lambda \vec{\beta} \cdot \vec{\beta}$)





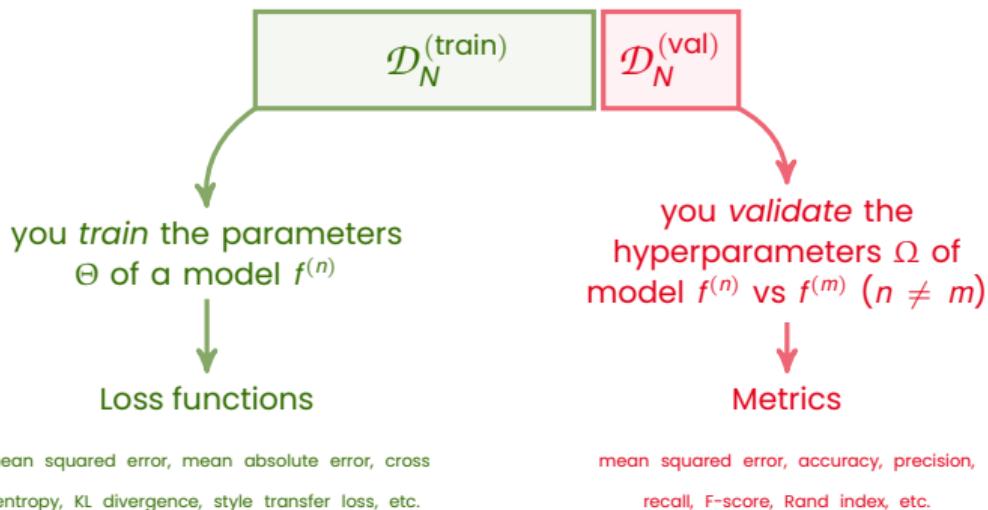
Validation

Parameters vs hyperparameters

Let $M = \{f^{(n)} \mid n = 1, 2, \dots\}$ set of *models* (e.g. linear model, support vector machine, decision tree, neural network, etc.)

Each $f^{(n)}$ has two sets of dependencies $\Rightarrow f^{(n)} = f^{(n)}(\Theta; \Omega)$:

- Θ is the set of **parameters** (i.e. the weights of the model $\rightarrow y = \vec{\beta} \cdot \vec{x}$)
- Ω is the set of **hyperparameters** (i.e. constraints of the model $\rightarrow y = \vec{\beta} \cdot \vec{x} + \lambda \vec{\beta} \cdot \vec{\beta}$)





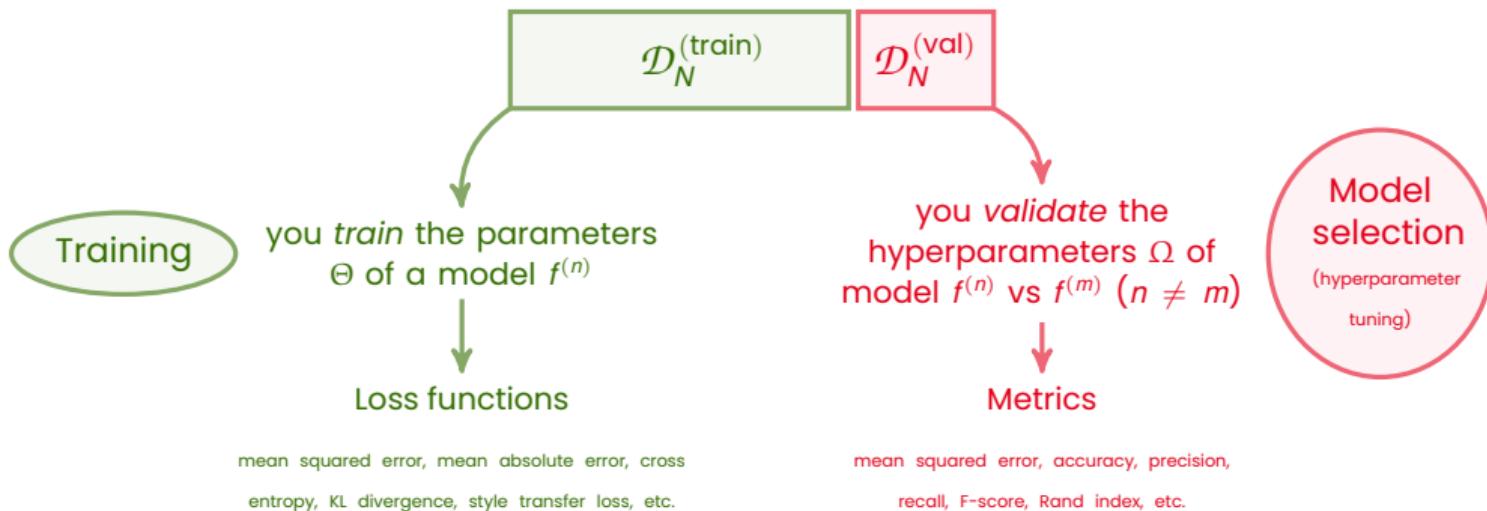
Validation

Parameters vs hyperparameters

Let $M = \{f^{(n)} \mid n = 1, 2, \dots\}$ set of *models* (e.g. linear model, support vector machine, decision tree, neural network, etc.)

Each $f^{(n)}$ has two sets of dependencies $\Rightarrow f^{(n)} = f^{(n)}(\Theta; \Omega)$:

- Θ is the set of **parameters** (i.e. the weights of the model $\rightarrow y = \vec{\beta} \cdot \vec{x}$)
- Ω is the set of **hyperparameters** (i.e. constraints of the model $\rightarrow y = \vec{\beta} \cdot \vec{x} + \lambda \vec{\beta} \cdot \vec{\beta}$)





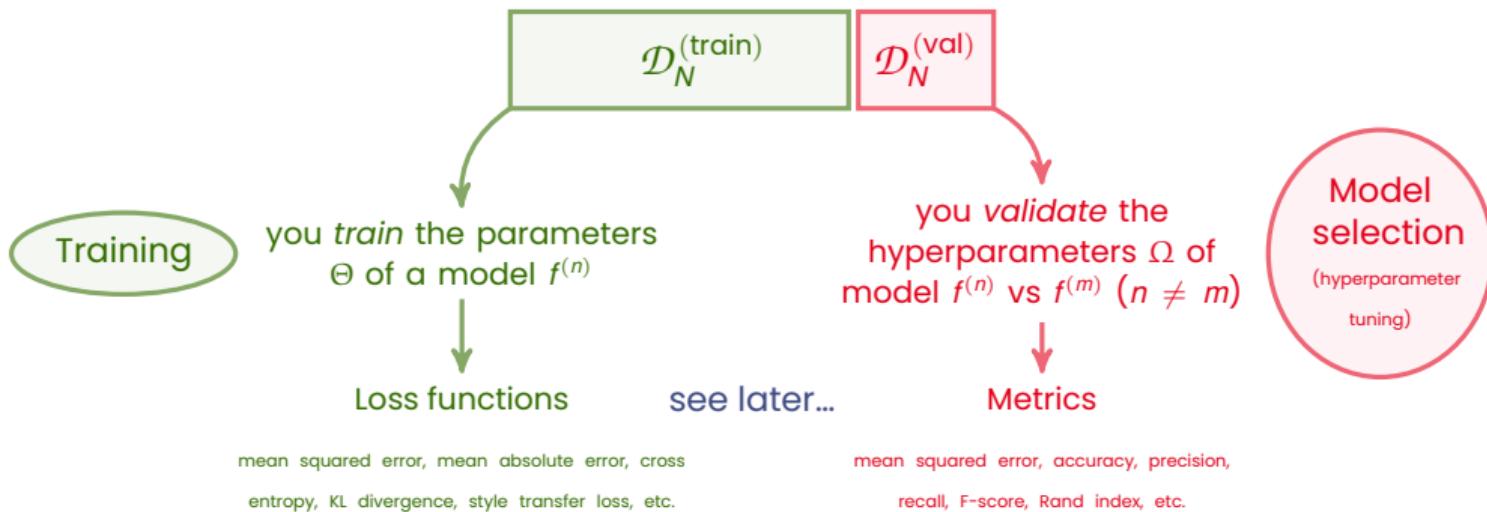
Validation

Parameters vs hyperparameters

Let $M = \{f^{(n)} \mid n = 1, 2, \dots\}$ set of *models* (e.g. linear model, support vector machine, decision tree, neural network, etc.)

Each $f^{(n)}$ has two sets of dependencies $\Rightarrow f^{(n)} = f^{(n)}(\Theta; \Omega)$:

- Θ is the set of **parameters** (i.e. the weights of the model $\rightarrow y = \vec{\beta} \cdot \vec{x}$)
- Ω is the set of **hyperparameters** (i.e. constraints of the model $\rightarrow y = \vec{\beta} \cdot \vec{x} + \lambda \vec{\beta} \cdot \vec{\beta}$)





Validation metrics

Evaluating a model

The plethora of evaluation functions at our disposal strongly depends on the task!



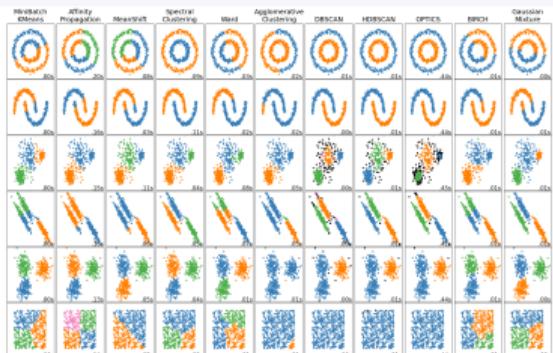
Validation metrics

Evaluating a model

The plethora of evaluation functions at our disposal strongly depends on the task!

Clustering task – Purity

Let $K = \{k_1, k_2, \dots, k_P\}$ the set of clusters, and $C = \{c_1, c_2, \dots, c_L\}$ the set of classes of N points:



$$P(K, C) = \frac{1}{N} \sum_{p=1}^P \max_{\ell=1, \dots, L} |k_p \cap c_\ell|$$

Purity is the normalised mode of the clusters. What happens if $K = N$?



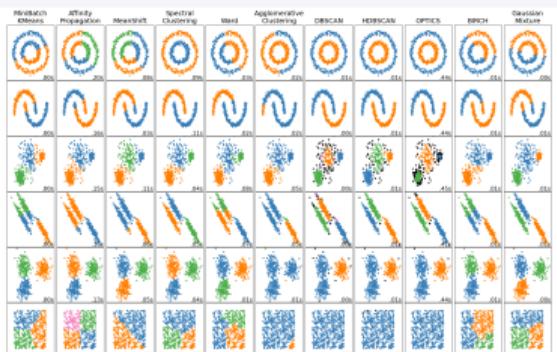
Validation metrics

Evaluating a model

The plethora of evaluation functions at our disposal strongly depends on the task!

Clustering task – Normalized Mutual Information

Let $K = \{k_1, k_2, \dots, k_P\}$ the set of clusters, and $C = \{c_1, c_2, \dots, c_L\}$ the set of classes of N points:



$$NMI(K, C) = 2 \frac{I(K, C)}{H(K) + H(C)}$$

where $H(\cdot) = -\mathbb{E}_{\mathcal{P}(\cdot)} [\ln \mathcal{P}(\cdot)]$ and

$$I(K, C) = \mathbb{E}_{\mathcal{P}(K \cap C)} \left[\ln \frac{\mathcal{P}(K \cap C)}{\mathcal{P}(K)\mathcal{P}(C)} \right]$$



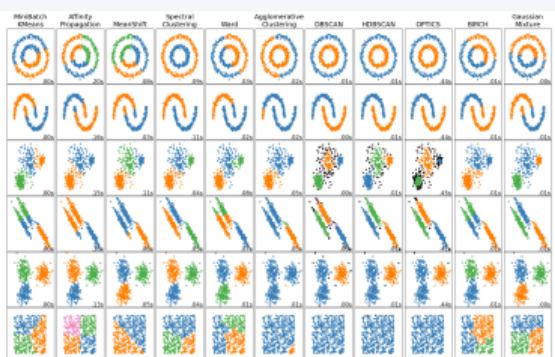
Validation metrics

Evaluating a model

The plethora of evaluation functions at our disposal strongly depends on the task!

Clustering task – Rand Index

Let $K = \{k_1, k_2, \dots, k_P\}$ the set of clusters, and $C = \{c_1, c_2, \dots, c_L\}$ the set of classes of N points:



Consider the $N(N - 1)/2$ couples:

- TP → similar objects in the same clusters
- TN → different objects in different clusters
- FP → *different* objects in the same clusters
- FN → *similar* objects in different clusters

$$RI(K, C) = \frac{TP + TN}{TP + TN + FP + FN}$$

Some of you might recognise the classification "accuracy" in this definition: even though the idea is not far, this is different!



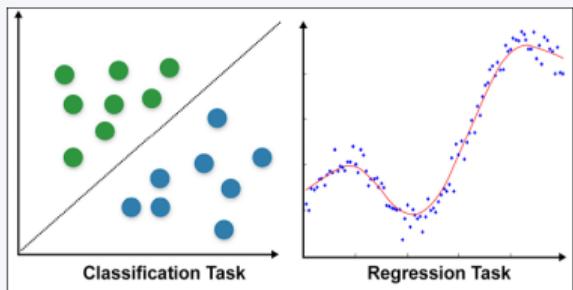
Validation metrics

Evaluating a model

The plethora of evaluation functions at our disposal strongly depends on the task!

Regression task – p-norm

Let $y_i \in \mathbb{R}$ be the *ground truth*, and $\hat{y}_i \in \mathbb{R}$ the *prediction* of the i -th sample ($i = 1, 2, \dots, N$).



$$\|y - \hat{y}\|_p = \left(\sum_{i=1}^N (y_i - \hat{y}_i)^p \right)^{\frac{1}{p}}$$

Specific cases:

- $p = 0 \rightarrow$ no. of non-zero elements
- $p = \infty \rightarrow$ can you compute it?



Validation metrics

Evaluating a model

The plethora of evaluation functions at our disposal strongly depends on the task!

Classification task – Accuracy, precision, recall and confusion matrix

Let $C_i \in \{0, 1\}$ be the *ground truth*, and $\hat{C}_i \in \{0, 1\}$ the *prediction* of the i -th sample ($i = 1, 2, \dots, N$).*

		prediction	
		$\hat{C} = 1$	$\hat{C} = 0$
ground truth	$C = 1$	TP	FN
	$C = 0$	FP	TN
confusion matrix			

Consider the possibilities:

- TP $\rightarrow C = \hat{C} = 1$
- TN $\rightarrow C = \hat{C} = 0$
- FP $\rightarrow C = 0$ and $\hat{C} = 1$ (type I)
- FN $\rightarrow C = 1$ and $\hat{C} = 0$ (type II)

* Class assignments are based on the *probability* of belonging to a class, that is $\hat{C} = 1 \Leftrightarrow P(\hat{Y} = 1) > \eta$, where η is an arbitrary threshold.



Validation metrics

Evaluating a model

The plethora of evaluation functions at our disposal strongly depends on the task!

Classification task – Accuracy, precision, recall and confusion matrix

Let $C_i \in \{0, 1\}$ be the *ground truth*, and $\hat{C}_i \in \{0, 1\}$ the *prediction* of the i -th sample ($i = 1, 2, \dots, N$).*

		prediction	
		$\hat{C} = 1$	$\hat{C} = 0$
ground truth	$C = 1$	TP	FN
	$C = 0$	FP	TN
confusion matrix			

$$\text{accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$

$$\text{precision} = \frac{TP}{TP+FP}$$

$$\text{recall} = \frac{TP}{TP+FN}$$

* Class assignments are based on the *probability* of belonging to a class, that is $\hat{C} = 1 \Leftrightarrow P(\hat{Y} = 1) > \eta$, where η is an arbitrary threshold.



Validation metrics

Evaluating a model

The plethora of evaluation functions at our disposal strongly depends on the task!

Classification task – Accuracy, precision, recall and confusion matrix

Let $C_i \in \{0, 1\}$ be the *ground truth*, and $\hat{C}_i \in \{0, 1\}$ the *prediction* of the i -th sample ($i = 1, 2, \dots, N$).*

		prediction	
		$\hat{C} = 1$	$\hat{C} = 0$
ground truth	$C = 1$	TP	FN
	$C = 0$	FP	TN
confusion matrix			

$$F_\beta = (1 + \beta^2) \frac{\text{precision} \cdot \text{recall}}{\beta^2 \cdot \text{precision} + \text{recall}}$$

Imagine you are testing the presence of an infection in the population: would you prefer a highly *precise* test or go for higher *recall*? Why?

* Class assignments are based on the *probability* of belonging to a class, that is $\hat{C} = 1 \Leftrightarrow P(\hat{Y} = 1) > \eta$, where η is an arbitrary threshold.



Validation metrics

Evaluating a model

The plethora of evaluation functions at our disposal strongly depends on the task!

Classification task – Accuracy, precision, recall and confusion matrix

Let $C_i \in \{0, 1\}$ be the *ground truth*, and $\hat{C}_i \in \{0, 1\}$ the *prediction* of the i -th sample ($i = 1, 2, \dots, N$).*

		prediction	
		$\hat{C} = 1$	$\hat{C} = 0$
ground truth	$C = 1$	TP	FN
	$C = 0$	FP	TN

$$\text{sensitivity} = \frac{TP}{TP+FN} \quad (= \text{recall})$$

$$\text{specificity} = \frac{TN}{TN+FP}$$

* Class assignments are based on the *probability* of belonging to a class, that is $\hat{C} = 1 \Leftrightarrow P(\hat{Y} = 1) > \eta$, where η is an arbitrary threshold.



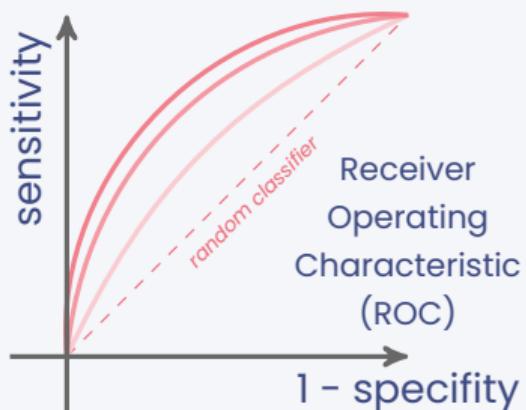
Validation metrics

Evaluating a model

The plethora of evaluation functions at our disposal strongly depends on the task!

Classification task – Accuracy, precision, recall and confusion matrix

Let $C_i \in \{0, 1\}$ be the *ground truth*, and $\hat{C}_i \in \{0, 1\}$ the *prediction* of the i -th sample ($i = 1, 2, \dots, N$).*



$$\text{sensitivity} = \frac{TP}{TP+FN} \quad (= \text{recall})$$

$$\text{specificity} = \frac{TN}{TN+FP}$$

All metrics depend on the **decision threshold** $M = M(\eta)$. What if we use it as a parameter? We can use the **Area Under the Curve** (AUC) to evaluate the classifier!

* Class assignments are based on the *probability* of belonging to a class, that is $\hat{C} = 1 \Leftrightarrow P(\hat{Y} = 1) > \eta$, where η is an arbitrary threshold.



Validation metrics

Evaluating a model

The plethora of evaluation functions at our disposal strongly depends on the task!

Binary classification

The output of the model for the i -th sample $\hat{y}_i \in \mathbb{R}^K$. We can use a **sigmoid** normalisation

$$\hat{y}'_i = \frac{1}{1 + e^{-\hat{y}_i}} \in [0, 1]$$

to **interpret** the result as a probability of belonging to the positive class. In other words, the class assignment is:

$$\hat{C} = 1 \Leftrightarrow P(Y_i = 1) = \hat{y}'_i > \eta,$$

where η is an arbitrary threshold (e.g. $\eta = 0.5$).



Validation metrics

Evaluating a model

The plethora of evaluation functions at our disposal strongly depends on the task!

Multiclass classification

The output of the model for the i -th sample $\hat{y}_i \in \mathbb{R}^K$. We can use a **softmax** normalisation

$$P(Y_i = k) = \hat{y}'^{(k)} = \frac{e^{\hat{y}_i^{(k)}}}{\sum_{\ell=1}^K e^{\hat{y}_i^{(\ell)}}} \quad \text{s.t.} \quad \sum_{k=1}^K \hat{y}'^{(k)} = 1$$

to **interpret** the result as a probability of belonging to the k -th class. In other words, the class assignment is:

$$\hat{C}_i = \arg \max_{k=1,\dots,K} \hat{y}'^{(k)} = \arg \max_{k=1,\dots,K} \hat{y}_i^{(k)}$$



Validation metrics

Evaluating a model

The plethora of evaluation functions at our disposal strongly depends on the task!

Target encoding

Compare **softmax**-activated layer classes \Rightarrow use **one-hot encoding**:

$$P(Y_i = k) = 1 \quad \Rightarrow \quad \vec{y}_i = \left(0, \dots, 0, \underbrace{1}_{k\text{-th position}}, 0, \dots, 0 \right).$$

```
 1 import numpy as np
 2
 3
 4 def one_hot_encoding(y: np.ndarray, n_classes: int) -> np.ndarray:
 5     n_samples = y.shape[0] # no. of samples in y
 6     one_hot = np.zeros((n_samples, n_classes)) # no. of classes
 7     one_hot[np.arange(n_samples), y] = 1 # set 1 depending on ground truth
 8
 9     return one_hot
```

This enables comparisons \vec{y} vs $\hat{\vec{y}}$ after **softmax** by comparing “bits” of information contained in the vectors.



2. The ML Mindset

- The variance vs bias trade-off

Table of contents

1. Some History and Philosophy to Start

2. The ML Mindset

Data preparation

Validation and test sets

The variance vs bias trade-off

Learning objectives and loss functions

Regularisation

3. ML Algorithms

4. Neural Networks

5. Conclusions





The ML Mindset

Variance and bias

Objective

Build a model which **learns** to **predict** (*generalise*) meaningful data

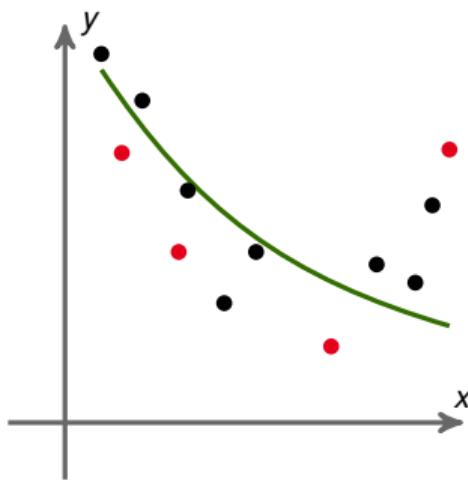
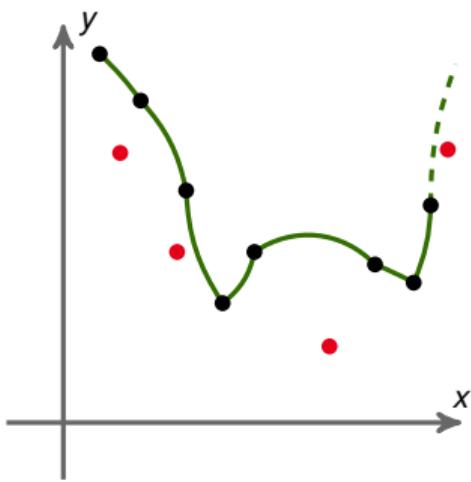


The ML Mindset

Variance and bias

Objective

Build a model which **learns** to **predict** (generalise) meaningful data





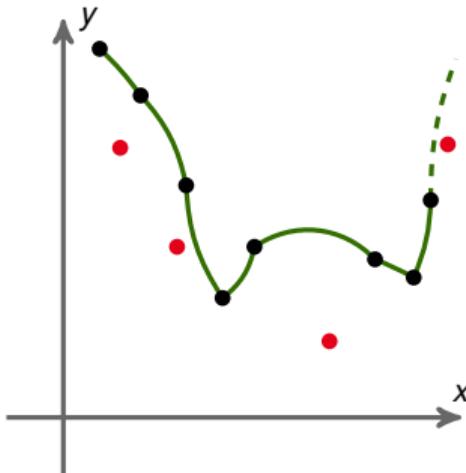
The ML Mindset

Variance and bias

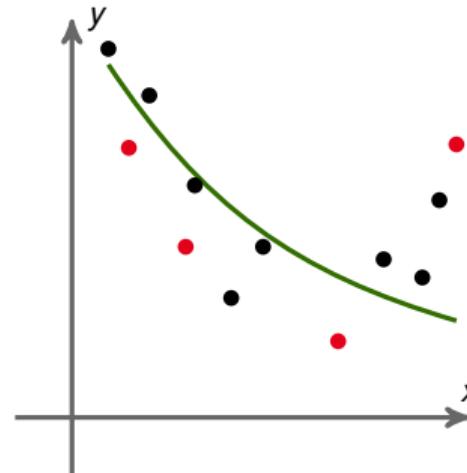
Objective

Build a model which **learns** to **predict** (generalise) meaningful data

Less assumptions more parameters



More assumptions less parameters





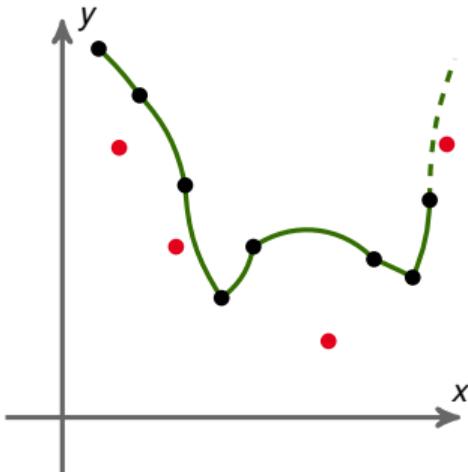
The ML Mindset

Variance and bias

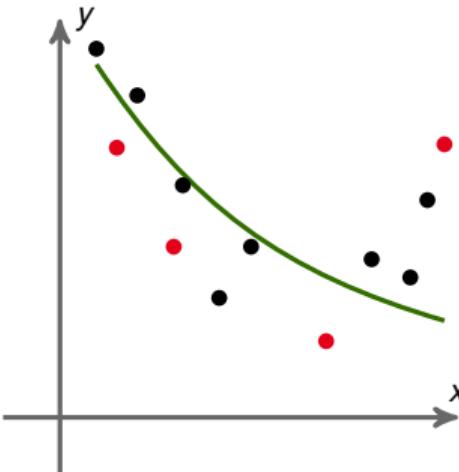
Objective

Build a model which **learns** to **predict** (generalise) meaningful data

OVERFITTING MODEL



UNDERFITTING MODEL





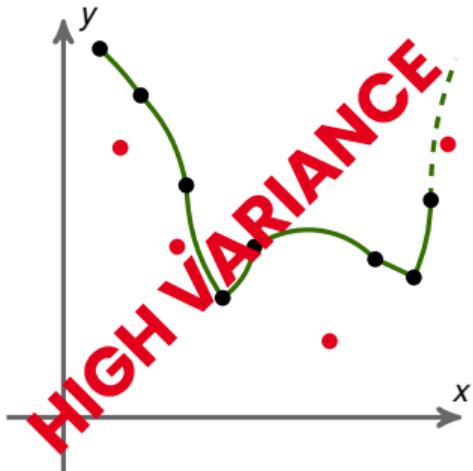
The ML Mindset

Variance and bias

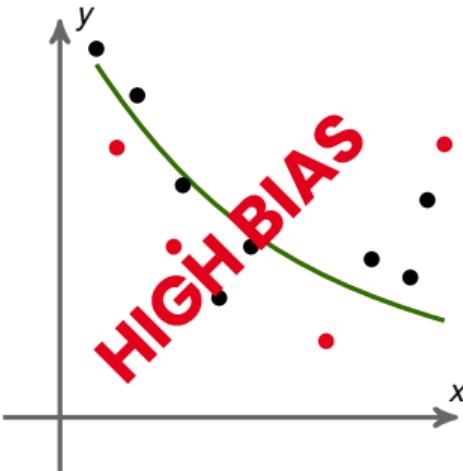
Objective

Build a model which **learns** to **predict** (generalise) meaningful data

OVERFITTING MODEL



UNDERFITTING MODEL





The Variance vs Bias Trade-off

Variance and bias in prediction

Without loss of generality, define the **true value**: N.B.: $\text{Var}(X) = \mathbb{E}[X^2] - \mathbb{E}[X]^2$

$$y = f(\vec{x}) + \varepsilon \in \mathbb{R}, \quad \vec{x} \in \mathbb{R}^p, \quad \mathbb{E}_{(\vec{x}, y)} [\varepsilon] = 0, \quad \text{Var}_{(\vec{x}, y)} (\varepsilon) = \mathbb{E}_{(\vec{x}, y)} [\varepsilon^2] = \sigma^2.$$



The Variance vs Bias Trade-off

Variance and bias in prediction

Without loss of generality, define the **true value**: N.B.: $\text{Var}(X) = \mathbb{E}[X^2] - \mathbb{E}[X]^2$

$$y = f(\vec{x}) + \varepsilon \in \mathbb{R}, \quad \vec{x} \in \mathbb{R}^p, \quad \mathbb{E}_{(\vec{x}, y)} [\varepsilon] = 0, \quad \text{Var}_{(\vec{x}, y)} (\varepsilon) = \mathbb{E}_{(\vec{x}, y)} [\varepsilon^2] = \sigma^2.$$

Build the **model** to predict a **label** (“supervised” see later)

$$\hat{f}_{\mathcal{D}_N}: \mathbb{R}^p \rightarrow \mathbb{R}$$

using data in $\mathcal{D}_N = \{(\vec{x}^{(i)}, y^{(i)}) \mid \vec{x}^{(i)} \in \mathbb{R}^p, y^{(i)} \in \mathbb{R} \quad \forall i = 1, 2, \dots, N\}$.



The Variance vs Bias Trade-off

Variance and bias in prediction

Without loss of generality, define the **true value**: N.B.: $\text{Var}(X) = \mathbb{E}[X^2] - \mathbb{E}[X]^2$

$$y = f(\vec{x}) + \varepsilon \in \mathbb{R}, \quad \vec{x} \in \mathbb{R}^p, \quad \mathbb{E}_{(\vec{x}, y)} [\varepsilon] = 0, \quad \text{Var}_{(\vec{x}, y)} (\varepsilon) = \mathbb{E}_{(\vec{x}, y)} [\varepsilon^2] = \sigma^2.$$

Build the **model** to predict a **label** (“supervised” see later)

$$\hat{f}_{\mathcal{D}_N}: \mathbb{R}^p \rightarrow \mathbb{R}$$

using data in $\mathcal{D}_N = \{(\vec{x}^{(i)}, y^{(i)}) \mid \vec{x}^{(i)} \in \mathbb{R}^p, y^{(i)} \in \mathbb{R} \quad \forall i = 1, 2, \dots, N\}$.

Generalisation error

Let (\vec{x}', y') be an **unseen** pair, and compute the squared error from a *trained* model $\hat{f}_{\mathcal{D}_N}$:

$$\mathbb{E}_{(\vec{x}, y), \mathcal{D}_N} \left[(y' - \hat{f}_{\mathcal{D}_N}(\vec{x}'))^2 \right]$$

We consider the *mean squared error* for simplicity, but the same holds for other kinds of generalisation error.



The Variance vs Bias Trade-off

Variance and bias in prediction

$$\mathbb{E}_{(\vec{x}, y), \mathcal{D}_N} \left[\left(y' - \hat{t}_{\mathcal{D}_N}(\vec{x}') \right)^2 \right]$$



The Variance vs Bias Trade-off

Variance and bias in prediction

$$\mathbb{E}_{(\vec{x}, y), \mathcal{D}_N} \left[(y' - \hat{t}_{\mathcal{D}_N}(\vec{x}'))^2 \right] = \mathbb{E}_{(\vec{x}, y), \mathcal{D}_N} \left[(\varepsilon + f(\vec{x}') - \hat{t}_{\mathcal{D}_N}(\vec{x}'))^2 \right]$$



The Variance vs Bias Trade-off

Variance and bias in prediction

$$\begin{aligned}\mathbb{E}_{(\vec{x}, y), \mathcal{D}_N} \left[\left(y' - \hat{f}_{\mathcal{D}_N}(\vec{x}') \right)^2 \right] &= \mathbb{E}_{(\vec{x}, y), \mathcal{D}_N} \left[\left(\varepsilon + f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}') \right)^2 \right] \\ &= \mathbb{E}_{(\vec{x}, y)} [\varepsilon^2] + \mathbb{E}_{\mathcal{D}_N} \left[\left(f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}') \right)^2 \right] + \mathbb{E}_{(\vec{x}, y), \mathcal{D}_N} \left[2 \varepsilon \left(f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}') \right) \right]\end{aligned}$$



The Variance vs Bias Trade-off

Variance and bias in prediction

$$\begin{aligned}\mathbb{E}_{(\vec{x}, y), \mathcal{D}_N} \left[(y' - \hat{f}_{\mathcal{D}_N}(\vec{x}'))^2 \right] &= \mathbb{E}_{(\vec{x}, y), \mathcal{D}_N} \left[(\varepsilon + f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}'))^2 \right] \\ &= \mathbb{E}_{(\vec{x}, y)} [\varepsilon^2] + \mathbb{E}_{\mathcal{D}_N} \left[(f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}'))^2 \right] + \mathbb{E}_{(\vec{x}, y), \mathcal{D}_N} \left[2 \varepsilon (f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}')) \right] \\ &= \sigma^2 + \mathbb{E}_{\mathcal{D}_N} \left[(f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}'))^2 \right] + 2 \mathbb{E}_{(\vec{x}, y)} [\varepsilon] \mathbb{E}_{\mathcal{D}_N} [f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}')]\end{aligned}$$



The Variance vs Bias Trade-off

Variance and bias in prediction

$$\begin{aligned}\mathbb{E}_{(\vec{x}, y), \mathcal{D}_N} \left[(y' - \hat{f}_{\mathcal{D}_N}(\vec{x}'))^2 \right] &= \mathbb{E}_{(\vec{x}, y), \mathcal{D}_N} \left[(\varepsilon + f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}'))^2 \right] \\ &= \mathbb{E}_{(\vec{x}, y)} [\varepsilon^2] + \mathbb{E}_{\mathcal{D}_N} \left[(f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}'))^2 \right] + \mathbb{E}_{(\vec{x}, y), \mathcal{D}_N} \left[2 \varepsilon (f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}')) \right] \\ &= \sigma^2 + \mathbb{E}_{\mathcal{D}_N} \left[(f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}'))^2 \right] + 2 \underbrace{\mathbb{E}_{(\vec{x}, y)} [\varepsilon]}_0 \mathbb{E}_{\mathcal{D}_N} \left[f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}') \right]\end{aligned}$$



The Variance vs Bias Trade-off

Variance and bias in prediction

$$\begin{aligned}\mathbb{E}_{(\vec{x},y), \mathcal{D}_N} \left[\left(y' - \hat{f}_{\mathcal{D}_N}(\vec{x}') \right)^2 \right] &= \mathbb{E}_{(\vec{x},y), \mathcal{D}_N} \left[\left(\varepsilon + f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}') \right)^2 \right] \\ &= \mathbb{E}_{(\vec{x},y)} [\varepsilon^2] + \mathbb{E}_{\mathcal{D}_N} \left[\left(f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}') \right)^2 \right] + \mathbb{E}_{(\vec{x},y), \mathcal{D}_N} \left[2 \varepsilon \left(f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}') \right) \right] \\ &= \sigma^2 + \mathbb{E}_{\mathcal{D}_N} \left[\left(f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}') \right)^2 \right] + 2 \cancel{\mathbb{E}_{(\vec{x},y)} [\varepsilon]}^0 \mathbb{E}_{\mathcal{D}_N} \left[f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}') \right] \\ &= \sigma^2 + \mathbb{E}_{\mathcal{D}_N} \left[f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}') \right]^2 + \text{Var}_{\mathcal{D}_N} \left(f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}') \right)\end{aligned}$$



The Variance vs Bias Trade-off

Variance and bias in prediction

$$\begin{aligned}\mathbb{E}_{(\vec{x}, y), \mathcal{D}_N} \left[(y' - \hat{f}_{\mathcal{D}_N}(\vec{x}'))^2 \right] &= \mathbb{E}_{(\vec{x}, y), \mathcal{D}_N} \left[(\varepsilon + f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}'))^2 \right] \\ &= \mathbb{E}_{(\vec{x}, y)} [\varepsilon^2] + \mathbb{E}_{\mathcal{D}_N} \left[(f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}'))^2 \right] + \mathbb{E}_{(\vec{x}, y), \mathcal{D}_N} \left[2\varepsilon (f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}')) \right] \\ &= \sigma^2 + \mathbb{E}_{\mathcal{D}_N} \left[(f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}'))^2 \right] + 2 \cancel{\mathbb{E}_{(\vec{x}, y)} [\varepsilon]}^0 \mathbb{E}_{\mathcal{D}_N} \left[f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}') \right] \\ &= \sigma^2 + \mathbb{E}_{\mathcal{D}_N} \left[f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}') \right]^2 + \text{Var}_{\mathcal{D}_N} \left(f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}') \right) \\ &= \sigma^2 + \mathbb{E}_{\mathcal{D}_N} \left[f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}') \right]^2 + \text{Var}_{\mathcal{D}_N} \left(\hat{f}_{\mathcal{D}_N}(\vec{x}') \right)\end{aligned}$$



The Variance vs Bias Trade-off

Variance and bias in prediction

$$\begin{aligned}\mathbb{E}_{(\vec{x}, y), \mathcal{D}_N} \left[(y' - \hat{f}_{\mathcal{D}_N}(\vec{x}'))^2 \right] &= \mathbb{E}_{(\vec{x}, y), \mathcal{D}_N} \left[(\varepsilon + f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}'))^2 \right] \\ &= \mathbb{E}_{(\vec{x}, y)} [\varepsilon^2] + \mathbb{E}_{\mathcal{D}_N} \left[(f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}'))^2 \right] + \mathbb{E}_{(\vec{x}, y), \mathcal{D}_N} \left[2\varepsilon (f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}')) \right] \\ &= \sigma^2 + \mathbb{E}_{\mathcal{D}_N} \left[(f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}'))^2 \right] + 2 \underbrace{\mathbb{E}_{(\vec{x}, y)} [\varepsilon]}_0 \mathbb{E}_{\mathcal{D}_N} [f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}')] \\ &= \sigma^2 + \mathbb{E}_{\mathcal{D}_N} \left[f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}') \right]^2 + \text{Var}_{\mathcal{D}_N} (f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}')) \\ &= \underbrace{\sigma^2}_{\text{irreducible error}} + \underbrace{\mathbb{E}_{\mathcal{D}_N} \left[f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}') \right]^2}_{\text{bias}^2} + \underbrace{\text{Var}_{\mathcal{D}_N} (\hat{f}_{\mathcal{D}_N}(\vec{x}'))}_{\text{variance}}\end{aligned}$$



The Variance vs Bias Trade-off

Variance and bias in prediction

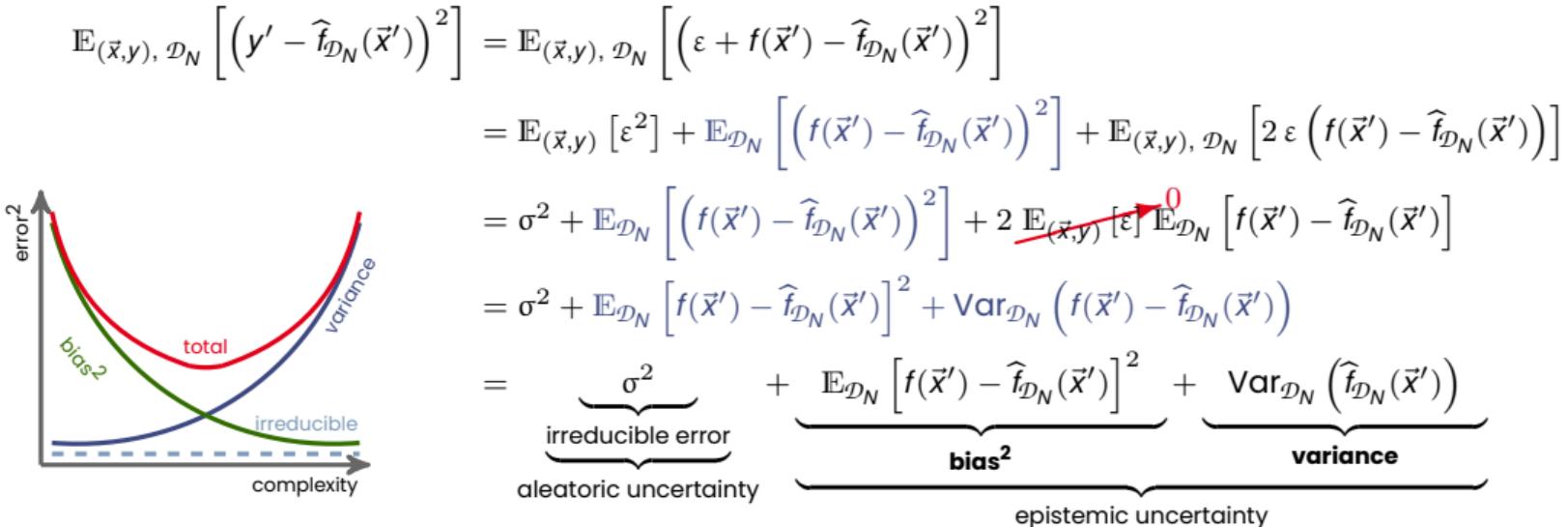
$$\begin{aligned}\mathbb{E}_{(\vec{x},y), \mathcal{D}_N} \left[\left(y' - \hat{f}_{\mathcal{D}_N}(\vec{x}') \right)^2 \right] &= \mathbb{E}_{(\vec{x},y), \mathcal{D}_N} \left[\left(\varepsilon + f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}') \right)^2 \right] \\&= \mathbb{E}_{(\vec{x},y)} [\varepsilon^2] + \mathbb{E}_{\mathcal{D}_N} \left[\left(f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}') \right)^2 \right] + \mathbb{E}_{(\vec{x},y), \mathcal{D}_N} \left[2\varepsilon \left(f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}') \right) \right] \\&= \sigma^2 + \mathbb{E}_{\mathcal{D}_N} \left[\left(f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}') \right)^2 \right] + 2 \underbrace{\mathbb{E}_{(\vec{x},y)} [\varepsilon]}_0 \mathbb{E}_{\mathcal{D}_N} \left[f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}') \right] \\&= \sigma^2 + \mathbb{E}_{\mathcal{D}_N} \left[f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}') \right]^2 + \text{Var}_{\mathcal{D}_N} \left(f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}') \right) \\&= \underbrace{\sigma^2}_{\substack{\text{irreducible error} \\ \text{aleatoric uncertainty}}} + \underbrace{\mathbb{E}_{\mathcal{D}_N} \left[f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}') \right]^2}_{\text{bias}^2} + \underbrace{\text{Var}_{\mathcal{D}_N} \left(\hat{f}_{\mathcal{D}_N}(\vec{x}') \right)}_{\substack{\text{variance} \\ \text{epistemic uncertainty}}}\end{aligned}$$

High bias or **high variance** produce high/bad **generalisation errors!**
The choice of a good **validation strategy** becomes **fundamental!**



The Variance vs Bias Trade-off

Variance and bias in prediction



High bias or high variance produce high/bad **generalisation errors!**
The choice of a good **validation strategy** becomes **fundamental!**



Back to Cross Validation

A variance and bias perspective

What K should I choose? What happens to the **estimate** of the **prediction error**

$$\mathcal{E}_{\mathcal{D}_N^{(\text{val})}}^{(K)}(y, \hat{y}) = \mathbb{E}_{\text{K-folds}} \left[\mathbb{E}_{\mathcal{D}_N^{(\text{val})}} [\text{dist}(y, \hat{y})] \right]$$

in **cross validation**?



Back to Cross Validation

A variance and bias perspective

What K should I choose? What happens to the **estimate** of the **prediction error** in **cross validation**?

Consider its **stability analysis**: (i.e. the study of its covariance, as it is an average of i.i.d. variables \Rightarrow central limit theorem. See

Bengio and Grandvalet, 2004)

$$\text{Var}_{\mathcal{D}_N^{(\text{val})}}^{(K)} (\text{dist}(y, \hat{y})) = \frac{1}{K^2} \sum_{i,j=0}^{K-1} \frac{1}{m_i m_j} \sum_{p,q=0}^{m_{ij}-1} \text{Cov} \left(\text{dist}(y_p^{(i)}, \hat{y}_p^{(i)}), \text{dist}(y_q^{(j)}, \hat{y}_q^{(j)}) \right)$$



Back to Cross Validation

A variance and bias perspective

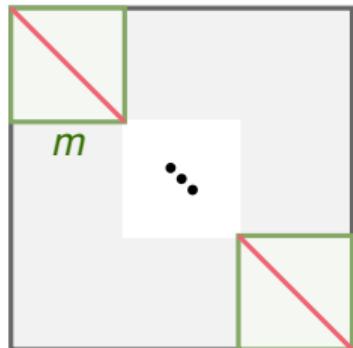
What K should I choose? What happens to the **estimate** of the **prediction error** in **cross validation**?

Consider its **stability analysis**: (i.e. the study of its covariance, as it is an average of i.i.d. variables \Rightarrow central limit theorem. See

Bengio and Grandvalet, 2004)

$$\text{Var}_{\mathcal{D}_N^{(\text{val})}}^{(K)} (\text{dist}(y, \hat{y})) = \frac{1}{K^2} \sum_{i,j=0}^{K-1} \frac{1}{m_i m_j} \sum_{p,q=0}^{m_{i,j}-1} \text{Cov} \left(\text{dist}(y_p^{(i)}, \hat{y}_p^{(i)}), \text{dist}(y_q^{(j)}, \hat{y}_q^{(j)}) \right)$$

$$= \sum_{\text{everything}}$$



$$= \frac{1}{n^2} \sigma^2 + \frac{m-1}{n} \omega + \frac{n-m}{n} \gamma$$

Lemma: \nexists unbiased estimator of $\text{Var}(\text{dist}(y, \hat{y}))$



Back to Cross Validation

A variance and bias perspective

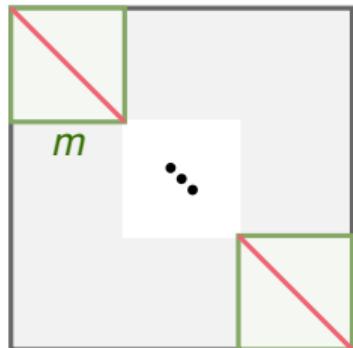
What K should I choose? What happens to the **estimate** of the **prediction error** in **cross validation**?

Consider its **stability analysis**: (i.e. the study of its covariance, as it is an average of i.i.d. variables \Rightarrow central limit theorem. See

Bengio and Grandvalet, 2004)

$$\text{Var}_{\mathcal{D}_N^{(\text{val})}}^{(K)} (\text{dist}(y, \hat{y})) = \frac{1}{K^2} \sum_{i,j=0}^{K-1} \frac{1}{m_i m_j} \sum_{p,q=0}^{m_{ij}-1} \text{Cov} \left(\text{dist}(y_p^{(i)}, \hat{y}_p^{(i)}), \text{dist}(y_q^{(j)}, \hat{y}_q^{(j)}) \right)$$

$$= \sum_{\text{everything}}$$



ω and γ depend on correlations!

$$= \frac{1}{n^2} \sigma^2 + \frac{m-1}{n} \omega + \frac{n-m}{n} \gamma$$

Lemma: \nexists unbiased estimator of $\text{Var}(\text{dist}(y, \hat{y}))$



2. The ML Mindset

- Learning objectives and loss functions

Table of contents

1. Some History and Philosophy to Start

2. The ML Mindset

Data preparation

Validation and test sets

The variance vs bias trade-off

Learning objectives and loss functions

Regularisation

3. ML Algorithms

4. Neural Networks

5. Conclusions





Learning Objectives

Do machines learn?

Consider all elements:

- a “machine” needs **good data** as input even though nobody wants to tidy data for life...
- we need **structured** procedures to avoid mistakes
- we must use **good practices** (data split, validation, etc.)
- we have to deal with **bias** and **variance**
- a “machine” needs an **architecture** and an **objective** to train what everyone wants!



Learning Objectives

Do machines learn?

Consider all elements:

- a “machine” needs **good data** as input even though nobody wants to tidy data for life...
- we need **structured** procedures to avoid mistakes
- we must use **good practices** (data split, validation, etc.)
- we have to deal with **bias** and **variance**
- a “machine” needs an **architecture** and an **objective** to train what everyone wants!

How does a “machine” learn?





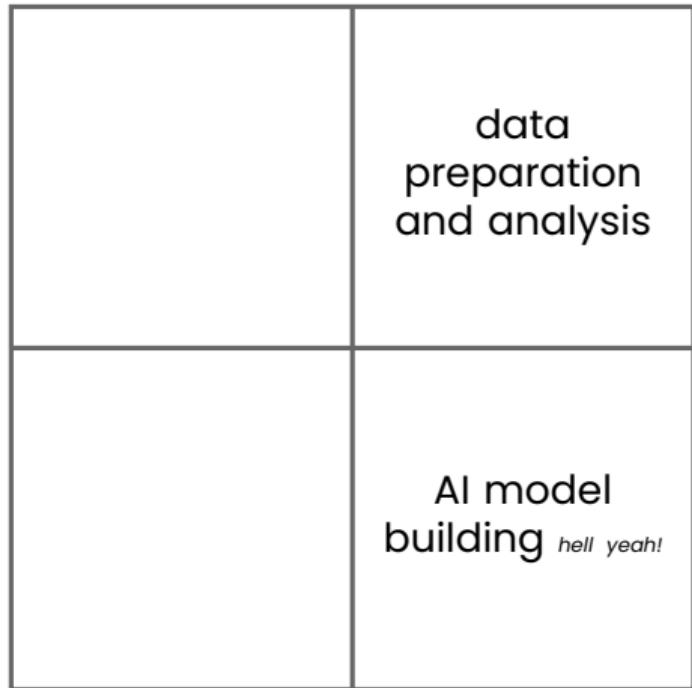
Learning Objectives

Do machines learn?

Consider all elements:

- a “machine” needs **good data** as input even though nobody wants to tidy data for life...
- we need **structured** procedures to avoid mistakes
- we must use **good practices** (data split, validation, etc.)
- we have to deal with **bias** and **variance**
- a “machine” needs an **architecture** and an **objective** to train *what everyone wants!*

How does a “machine” learn?





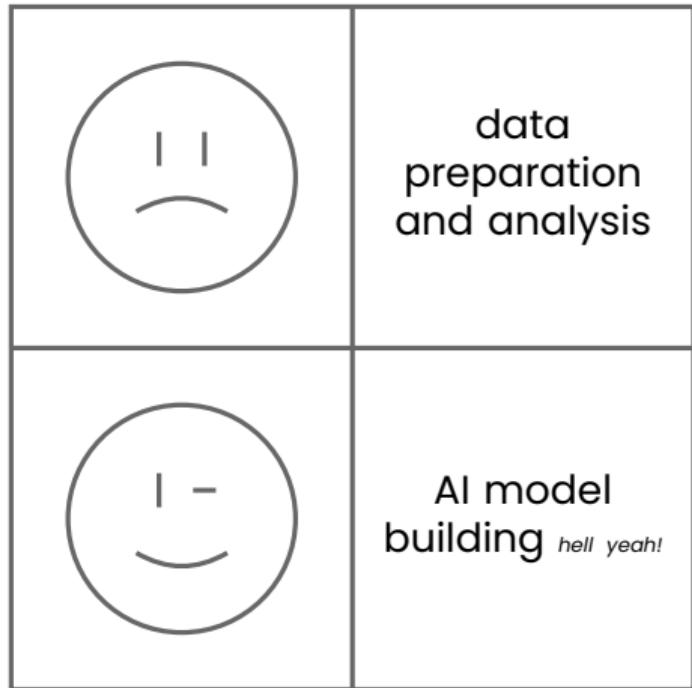
Learning Objectives

Do machines learn?

Consider all elements:

- a “machine” needs **good data** as input even though nobody wants to tidy data for life...
- we need **structured** procedures to avoid mistakes
- we must use **good practices** (data split, validation, etc.)
- we have to deal with **bias** and **variance**
- a “machine” needs an **architecture** and an **objective** to train *what everyone wants!*

How does a “machine” learn?





Learning Objectives

Do machines learn?

Consider all elements:

- a “machine” needs **good data** as input even though nobody wants to tidy data for life...
- we need **structured** procedures to avoid mistakes
- we must use **good practices** (data split, validation, etc.)
- we have to deal with **bias** and **variance**
- a “machine” needs an **architecture** and an **objective** to train what everyone wants!

How does a “machine” learn?





Learning Objectives

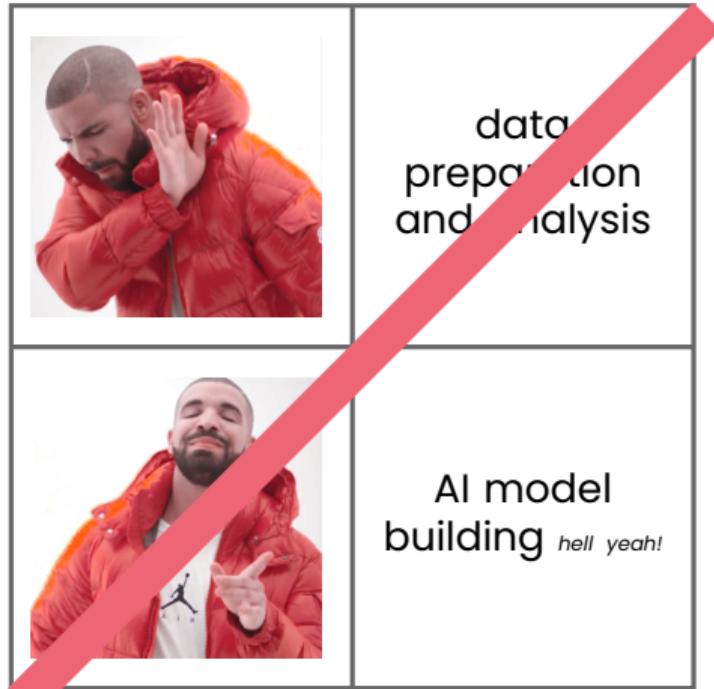
Do machines learn?

Consider all elements:

- a “machine” needs **good data** as input even though nobody wants to tidy data for life...
- we need **structured** procedures to avoid mistakes
- we must use **good practices** (data split, validation, etc.)
- we have to deal with **bias** and **variance**
- a “machine” needs an **architecture** and an **objective** to train *what everyone wants!*

How does a “machine” learn?

please, do not follow the advice, this is just a meme...





Loss Functions

Prediction estimation

Machines can learn in different ways:

$\text{dist}(y, \hat{y}) \xrightarrow{\text{better...}} (\mathcal{L}, \mathcal{K}, g)$ “**loss (function)**” (sometimes *Lagrangian*),

where $\mathcal{K} \sim \mathbb{C}^n$ (at least locally) with a **metric** tensor g :

$$\mathcal{L}: \mathcal{K} \longrightarrow \mathbb{R}$$

$$Z \longmapsto \mathcal{L}(Z) \stackrel{\text{def}}{=} \text{“distance from target”}$$



Loss Functions

Prediction estimation

Machines can learn in different ways:

$\text{dist}(y, \hat{y}) \xrightarrow{\text{better...}} (\mathcal{L}, \mathcal{K}, g)$ “**loss (function)**” (sometimes *Lagrangian*),

where $\mathcal{K} \sim \mathbb{C}^n$ (at least locally) with a **metric** tensor g :

$$\mathcal{L}: \mathcal{K} \longrightarrow \mathbb{R}$$

$$Z \longmapsto \mathcal{L}(Z) \stackrel{\text{def}}{=} \text{“distance from target”}$$

Notice $Z = Z(y, \hat{y})$ and $\hat{y} = f_{\{\Theta; \Omega\}}(x)$. The *training* problem (i.e. finding the best Θ^*) becomes:

$$\Theta^* = \arg \min_{\Theta} \mathcal{L}(Z) = \arg \min_{\Theta} \mathcal{L}(y, \hat{y}(\Theta, \Omega))$$

Should you see a correlation between \mathcal{L} and the logarithm of a **likelihood** function, you would be basically right...



Loss Functions

Prediction estimation

Machines can learn in different ways:

$\text{dist}(y, \hat{y}) \xrightarrow{\text{better...}} (\mathcal{L}, \mathcal{K}, g)$ “**loss (function)**” (sometimes *Lagrangian*),

where $\mathcal{K} \sim \mathbb{C}^n$ (at least locally) with a **metric** tensor g :

$$\mathcal{L}: \mathcal{K} \longrightarrow \mathbb{R}$$

$$Z \longmapsto \mathcal{L}(Z) \stackrel{\text{def}}{=} \text{“distance from target”}$$

Least squares

Let $Z = Y - \hat{Y}$:

$$\mathcal{L}(Z) = \mathbb{E}_{\mathcal{P}(Y)} \left[(Y - \hat{Y})^2 \right]$$

Cross entropy

Let $Z = (Y, \hat{Y})$:

$$\mathcal{L}(Z) = -\mathbb{E}_{\mathcal{P}(Y)} [\ln \hat{Y}]$$

K-means clustering

Let $Z = X - \hat{M}_c$:

$$\mathcal{L}(Z) = \mathbb{E}_{\mathcal{P}(X,C)} \left[(X - \hat{M}_c)^2 \right]$$



Loss Functions

Some properties

What makes a **loss function** “good”? These properties are not always verified unfortunately...

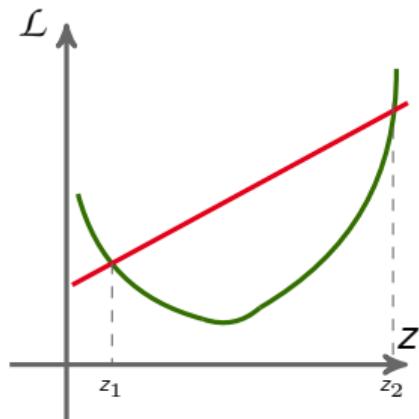


Loss Functions

Some properties

What makes a **loss function** “good”? These properties are not always verified unfortunately...

Convexity



$$\begin{aligned}\mathcal{L}(t z_1 + (1 - t) z_2) &\leq t \mathcal{L}(z_1) + (1 - t) \mathcal{L}(z_2) \\ \forall t &\in [0, 1]\end{aligned}$$

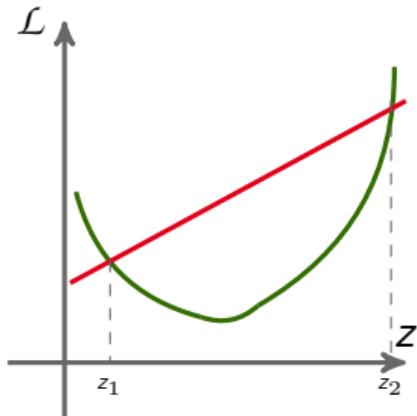


Loss Functions

Some properties

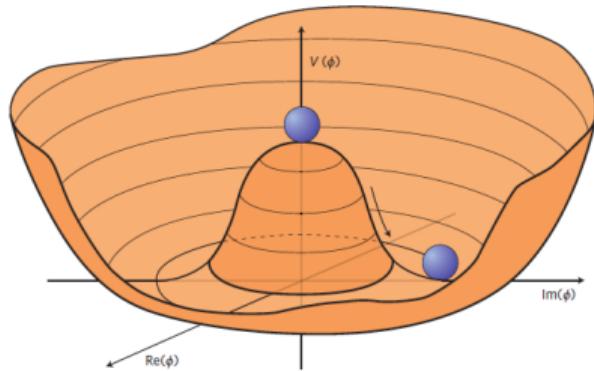
What makes a **loss function** “good”? These properties are not always verified unfortunately...

Convexity



$$\mathcal{L}(t z_1 + (1 - t) z_2) \leq t \mathcal{L}(z_1) + (1 - t) \mathcal{L}(z_2)$$
$$\forall t \in [0, 1]$$

Differentiability



$$\forall c \in \mathcal{K} \setminus D_0 \exists \lim_{x \rightarrow c} \frac{f(x) - f(c)}{x - c} = f'(x)|_{x=c}$$

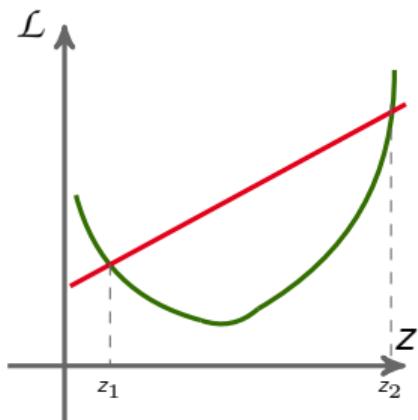


Loss Functions

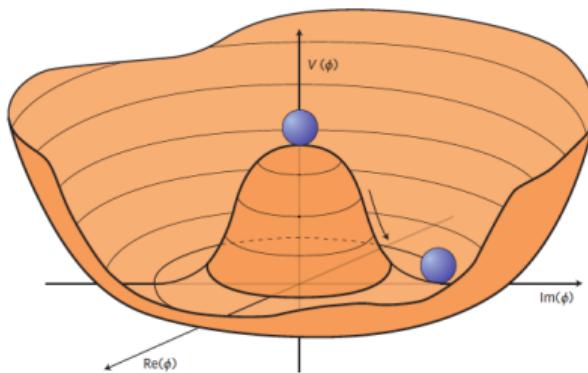
Some properties

What makes a **loss function** “good”? These properties are not always verified unfortunately...

Convexity



Differentiability



$$\mathcal{L}(z_2) - \mathcal{L}(z_1) \geq \frac{d\mathcal{L}(z)}{dz} \Big|_{z=z_1} (z_2 - z_1)$$

$$\forall c \in \mathcal{K} \setminus D_0 \exists \lim_{x \rightarrow c} \frac{f(x) - f(c)}{x - c} = f'(x)|_{x=c}$$



Loss Functions

An example of convex loss

Let \mathcal{P} and \mathcal{Q} be two probability distributions of $X \in \mathfrak{X}$, and consider the **Kullback-Leibler** divergence:

$$D_{KL}(\mathcal{P} \parallel \mathcal{Q}) = \mathbb{E}_{X \sim \mathcal{P}} \left[\ln \frac{\mathcal{P}(X)}{\mathcal{Q}(X)} \right] = \sum_{x \in \mathfrak{X}} \mathcal{P}(x) \ln \frac{\mathcal{P}(x)}{\mathcal{Q}(x)}.$$



Loss Functions

An example of convex loss

Let \mathcal{P} and \mathcal{Q} be two probability distributions of $X \in \mathfrak{X}$, and consider the **Kullback-Leibler** divergence:

$$D_{KL}(\mathcal{P} \parallel \mathcal{Q}) = \mathbb{E}_{X \sim \mathcal{P}} \left[\ln \frac{\mathcal{P}(X)}{\mathcal{Q}(X)} \right] = \sum_{x \in \mathfrak{X}} \mathcal{P}(x) \ln \frac{\mathcal{P}(x)}{\mathcal{Q}(x)}.$$

Theorem

$D_{KL}(\mathcal{P} \parallel \mathcal{Q})$ is a **convex** function in the pair $(\mathcal{P}, \mathcal{Q})$ over \mathfrak{X} .



Loss Functions

An example of convex loss

Let \mathcal{P} and \mathcal{Q} be two probability distributions of $X \in \mathfrak{X}$, and consider the **Kullback-Leibler** divergence:

$$D_{KL}(\mathcal{P} \parallel \mathcal{Q}) = \mathbb{E}_{X \sim \mathcal{P}} \left[\ln \frac{\mathcal{P}(X)}{\mathcal{Q}(X)} \right] = \sum_{x \in \mathfrak{X}} \mathcal{P}(x) \ln \frac{\mathcal{P}(x)}{\mathcal{Q}(x)}.$$

Theorem

$D_{KL}(\mathcal{P} \parallel \mathcal{Q})$ is a **convex** function in the pair $(\mathcal{P}, \mathcal{Q})$ over \mathfrak{X} .

Proof

$$\begin{aligned} D_{KL}(t \mathcal{P}_1(X) + (1-t) \mathcal{P}_2(X) \parallel t \mathcal{Q}_1(X) + (1-t) \mathcal{Q}_2(X)) &= \\ &= \sum_{x \in \mathfrak{X}} \left((t \mathcal{P}_1(X) + (1-t) \mathcal{P}_2(X)) \ln \frac{t \mathcal{P}_1(X) + (1-t) \mathcal{P}_2(X)}{t \mathcal{Q}_1(X) + (1-t) \mathcal{Q}_2(X)} \right) \end{aligned}$$



Loss Functions

An example of convex loss

Let \mathcal{P} and \mathcal{Q} be two probability distributions of $X \in \mathfrak{X}$, and consider the **Kullback-Leibler** divergence:

$$D_{KL}(\mathcal{P} \parallel \mathcal{Q}) = \mathbb{E}_{X \sim \mathcal{P}} \left[\ln \frac{\mathcal{P}(X)}{\mathcal{Q}(X)} \right] = \sum_{x \in \mathfrak{X}} \mathcal{P}(x) \ln \frac{\mathcal{P}(x)}{\mathcal{Q}(x)}.$$

Theorem

$D_{KL}(\mathcal{P} \parallel \mathcal{Q})$ is a **convex** function in the pair $(\mathcal{P}, \mathcal{Q})$ over \mathfrak{X} .

Proof

$$\begin{aligned} D_{KL}(t\mathcal{P}_1(X) + (1-t)\mathcal{P}_2(X) \parallel t\mathcal{Q}_1(X) + (1-t)\mathcal{Q}_2(X)) &\leq \\ \text{"log sum inequality"} &\leq \sum_{x \in \mathfrak{X}} \left(t\mathcal{P}_1(X) \ln \frac{t\mathcal{P}_1(X)}{t\mathcal{Q}_1(X)} + (1-t)\mathcal{P}_2(X) \ln \frac{(1-t)\mathcal{P}_2(X)}{(1-t)\mathcal{Q}_2(X)} \right) \end{aligned}$$



Loss Functions

An example of convex loss

Let \mathcal{P} and \mathcal{Q} be two probability distributions of $X \in \mathfrak{X}$, and consider the **Kullback-Leibler** divergence:

$$D_{KL}(\mathcal{P} \parallel \mathcal{Q}) = \mathbb{E}_{X \sim \mathcal{P}} \left[\ln \frac{\mathcal{P}(X)}{\mathcal{Q}(X)} \right] = \sum_{x \in \mathfrak{X}} \mathcal{P}(x) \ln \frac{\mathcal{P}(x)}{\mathcal{Q}(x)}.$$

Theorem

$D_{KL}(\mathcal{P} \parallel \mathcal{Q})$ is a **convex** function in the pair $(\mathcal{P}, \mathcal{Q})$ over \mathfrak{X} .

Proof

$$\begin{aligned} D_{KL}(t\mathcal{P}_1(X) + (1-t)\mathcal{P}_2(X) \parallel t\mathcal{Q}_1(X) + (1-t)\mathcal{Q}_2(X)) &\leq \\ &\leq t D_{KL}(\mathcal{P}_1 \parallel \mathcal{Q}_1) + (1-t) D_{KL}(\mathcal{P}_2 \parallel \mathcal{Q}_2) \end{aligned}$$

□



Loss Functions

An example of convex loss

Let \mathcal{P} and \mathcal{Q} be two probability distributions of $X \in \mathfrak{X}$, and consider the **Kullback-Leibler** divergence:

$$D_{KL}(\mathcal{P} \parallel \mathcal{Q}) = \mathbb{E}_{X \sim \mathcal{P}} \left[\ln \frac{\mathcal{P}(X)}{\mathcal{Q}(X)} \right] = \sum_{x \in \mathfrak{X}} \mathcal{P}(x) \ln \frac{\mathcal{P}(x)}{\mathcal{Q}(x)}.$$

Theorem

$D_{KL}(\mathcal{P} \parallel \mathcal{Q})$ is a **convex** function in the pair $(\mathcal{P}, \mathcal{Q})$ over \mathfrak{X} .

HOMEWORK

1. prove that $f(x) = -\ln(x)$ is **convex** (or that $f(x) = \ln(x)$ is **concave**)
2. prove the "log sum inequality" (it follows from **Jensen's inequality** and 1.) used in the proof – have fun or look it up!
3. prove that the cross entropy $H(\mathcal{P}, \mathcal{Q}) = -\mathbb{E}_{\mathcal{P}} [\ln \mathcal{Q}]$ is **convex** in \mathcal{Q} over \mathfrak{X}
4. prove that any *local minimum* of a convex function is also a *global minimum* (suppose there are more, and find a contradiction...)



Loss Functions

Differentiability and gradient descent

Remember the *minimisation* problem:

$$\Theta^* = \arg \min_{\Theta} \mathcal{L}(Z) = \arg \min_{\Theta} \mathcal{L}(y, \hat{y}(\Theta, \Omega))$$

What if \mathcal{L} is too complicated for an analytical solution?



Loss Functions

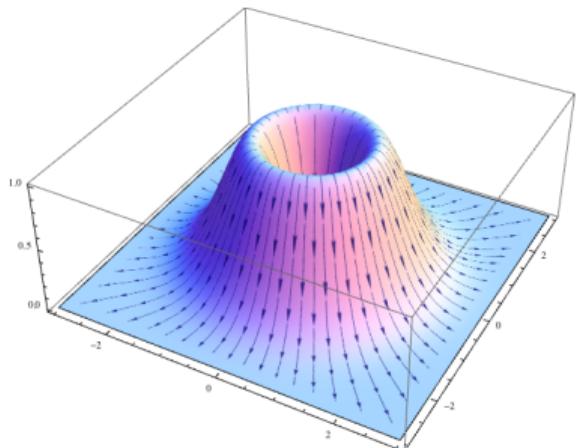
Differentiability and gradient descent

Remember the *minimisation* problem:

$$\Theta^* = \arg \min_{\Theta} \mathcal{L}(Z) = \arg \min_{\Theta} \mathcal{L}(y, \hat{y}(\Theta, \Omega))$$

What if \mathcal{L} is too complicated for an analytical solution?

Let $\vec{x}, \vec{v} \in \mathbb{R}^n$ and $f = f(\vec{x}) \in \mathcal{C}^2(\mathbb{R})$:



$$\vec{\nabla}_{\vec{v}} f(\vec{x}) = \vec{\nabla} f(\vec{x}) \cdot \vec{v}$$

which is maximal when \vec{v} is in the **same direction** of $\vec{\nabla} f(\vec{x})$

Steepest ascent

The **gradient** is the direction of **steepest ascent** of the (hyper)surface.



Loss Functions

Differentiability and gradient descent

Remember the *minimisation* problem:

$$\Theta^* = \arg \min_{\Theta} \mathcal{L}(Z) = \arg \min_{\Theta} \mathcal{L}(y, \hat{y}(\Theta, \Omega))$$

What if \mathcal{L} is too complicated for an analytical solution?

We can control the **descent** along the surface by iterating:

$$\vec{x}^{(t+1)} = \vec{x}^{(t)} - \alpha \odot \vec{\nabla} f \left(\vec{x}^{(t)} \right)$$

where $\alpha \in \mathbb{R}^n$ is the **learning rate** (N.B. $\alpha \in \Omega$ is a **hyperparameter** of the model often just a scalar...).



Loss Functions

Differentiability and gradient descent

Remember the *minimisation* problem:

$$\Theta^* = \arg \min_{\Theta} \mathcal{L}(Z) = \arg \min_{\Theta} \mathcal{L}(y, \hat{y}(\Theta, \Omega))$$

What if \mathcal{L} is too complicated for an analytical solution?

Gradient descent

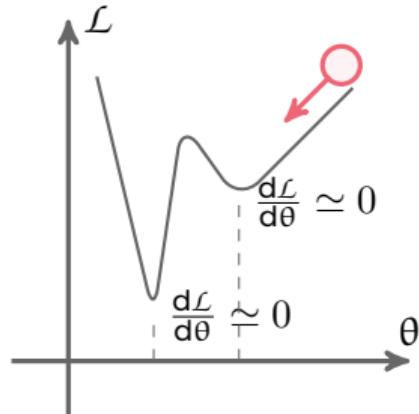
Require: $\alpha \in \mathbb{R}^+, \theta^{(0)}, \mathcal{L}, T \in \mathbb{N} \setminus \{0\}$

for $0 \leq t < T$ **do**

$$\vec{G}^{(t)} \leftarrow \vec{\nabla} \mathcal{L}(y, \hat{y}(\vec{\theta}^{(t)})) = \vec{\nabla} \mathcal{L}(\vec{\theta}^{(t)})$$

$$\vec{\theta}^{(t+1)} \leftarrow \vec{\theta}^{(t)} - \alpha \vec{G}^{(t)} \quad \triangleright \text{steepest descent}$$

return $\vec{\theta}^{(T)}$





Loss Functions

Differentiability and gradient descent

Remember the *minimisation* problem:

$$\Theta^* = \arg \min_{\Theta} \mathcal{L}(Z) = \arg \min_{\Theta} \mathcal{L}(y, \hat{y}(\Theta, \Omega))$$

What if \mathcal{L} is too complicated for an analytical solution?

Gradient descent

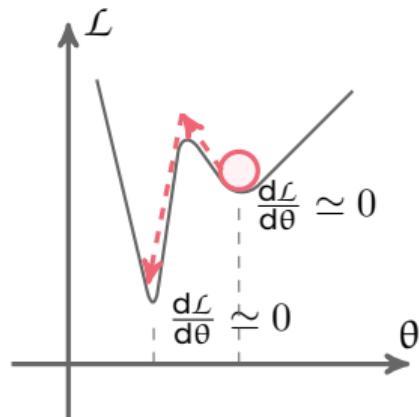
Require: $\alpha \in \mathbb{R}^+, \theta^{(0)}, \mathcal{L}, T \in \mathbb{N} \setminus \{0\}$

for $0 \leq t < T$ **do**

$$\vec{G}^{(t)} \leftarrow \vec{\nabla} \mathcal{L}(y, \hat{y}(\vec{\theta}^{(t)})) = \vec{\nabla} \mathcal{L}(\vec{\theta}^{(t)})$$

$$\vec{\theta}^{(t+1)} \leftarrow \vec{\theta}^{(t)} - \alpha \vec{G}^{(t)} \quad \triangleright \text{steepest descent}$$

return $\vec{\theta}^{(T)}$





Loss Functions

Differentiability and gradient descent

Remember the *minimisation* problem:

$$\Theta^* = \arg \min_{\Theta} \mathcal{L}(Z) = \arg \min_{\Theta} \mathcal{L}(y, \hat{y}(\Theta, \Omega))$$

What if \mathcal{L} is too complicated for an analytical solution?

Gradient descent

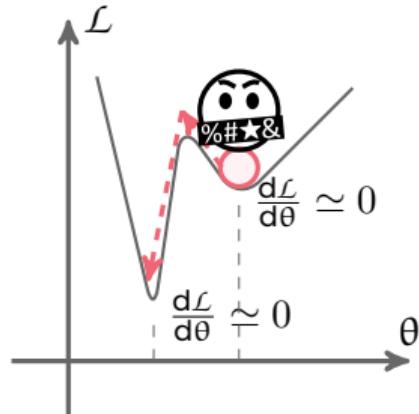
Require: $\alpha \in \mathbb{R}^+, \theta^{(0)}, \mathcal{L}, T \in \mathbb{N} \setminus \{0\}$

for $0 \leq t < T$ **do**

$$\vec{G}^{(t)} \leftarrow \vec{\nabla} \mathcal{L}(y, \hat{y}(\vec{\theta}^{(t)})) = \vec{\nabla} \mathcal{L}(\vec{\theta}^{(t)})$$

$$\vec{\theta}^{(t+1)} \leftarrow \vec{\theta}^{(t)} - \alpha \vec{G}^{(t)} \quad \triangleright \text{steepest descent}$$

return $\vec{\theta}^{(T)}$





Loss Functions

Differentiability and gradient descent

Remember the *minimisation* problem:

$$\Theta^* = \arg \min_{\Theta} \mathcal{L}(Z) = \arg \min_{\Theta} \mathcal{L}(y, \hat{y}(\Theta, \Omega))$$

What if \mathcal{L} is too complicated for an analytical solution?

Gradient descent

Require: $\alpha \in \mathbb{R}^+, \theta^{(0)}, \mathcal{L}, T \in \mathbb{N} \setminus \{0\}$

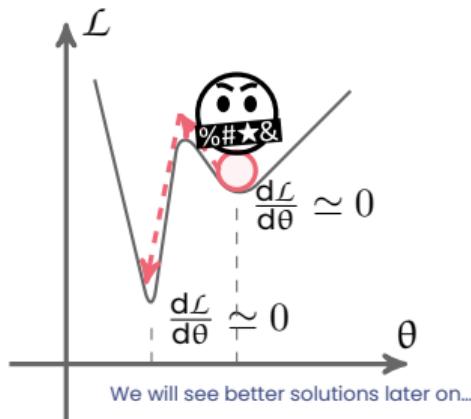
for $0 \leq t < T$ **do**

$$\vec{G}^{(t)} \leftarrow \vec{\nabla} \mathcal{L}(y, \hat{y}(\vec{\theta}^{(t)})) = \vec{\nabla} \mathcal{L}(\vec{\theta}^{(t)})$$

$$\vec{\theta}^{(t+1)} \leftarrow \vec{\theta}^{(t)} - \alpha \vec{G}^{(t)} \quad \triangleright \text{steepest descent}$$

return $\vec{\theta}^{(T)}$

Does it converge?





Gradient Descent

On the convergence of gradient descent

Definition | Lipschitz smoothness

Let $f \in \mathcal{C}^1(\mathbb{R}^n)$ a scalar function, and $L > 0$. We call f *L-smooth* if it is L -Lipschitz:

$$\forall \vec{x}, \vec{y} \in \mathbb{R}^n \quad \left\| \vec{\nabla}f(\vec{x}) - \vec{\nabla}f(\vec{y}) \right\|_2 \leq L \|\vec{x} - \vec{y}\|_2$$



Gradient Descent

On the convergence of gradient descent

Theorem | Smooth convex functions

Let f be a convex L -smooth scalar function over \mathbb{R}^n , and let $\alpha = L^{-1}$ the learning rate, then $\forall t \in [1, T]$:

$$f(\vec{x}^{(t)}) - f(\vec{x}^*) \leq \frac{2L}{T-1} \left\| \vec{x}^{(0)} - \vec{x}^* \right\|_2.$$

(see Gower (Télécom Paris))



Gradient Descent

On the convergence of gradient descent

Theorem | Smooth convex functions

Let f be a convex L -smooth scalar function over \mathbb{R}^n , and let $\alpha = L^{-1}$ the learning rate, then $\forall t \in [1, T]$:

$$f(\vec{x}^{(t)}) - f(\vec{x}^*) \leq \frac{2L}{T-1} \left\| \vec{x}^{(0)} - \vec{x}^* \right\|_2.$$

(see Gower (Télécom Paris))

Consider $\vec{x}^{(t+1)} = \vec{x}^{(t)} - \frac{1}{L} \vec{\nabla} f(\vec{x}^{(t)})$:

$$\left\| \vec{\nabla} f(\vec{x}^{(t+1)}) - \vec{\nabla} f(\vec{x}^{(t)}) \right\|_2 = \left\| \int_{\vec{x}^{(t)}}^{\vec{x}^{(t+1)}} d\vec{s} \, \nabla^2 f(\vec{s}) \right\|_2 \leq L \left\| \vec{x}^{(t+1)} - \vec{x}^{(t)} \right\|_2 = \frac{1}{L} \left\| \vec{\nabla} f(\vec{x}^{(t)}) \right\|_2$$

which shows (Taylor expansion + bound on Hessian) that

$$f(\vec{x}^{(t+1)}) \leq f(\vec{x}^{(t)}) - \frac{1}{L} \left\| \vec{\nabla} f(\vec{x}^{(t)}) \right\|_2 + \frac{1}{2L} \left\| \vec{\nabla} f(\vec{x}^{(t)}) \right\|_2 = f(\vec{x}^{(t)}) - \frac{1}{2L} \left\| \vec{\nabla} f(\vec{x}^{(t)}) \right\|_2.$$

Moreover, by subtracting $f(\vec{x}^*)$:

$$f(\vec{x}^{(t+1)}) - f(\vec{x}^*) \leq f(\vec{x}^{(t)}) - f(\vec{x}^*) - \frac{1}{2L} \left\| \vec{\nabla} f(\vec{x}^{(t)}) \right\|_2.$$



Gradient Descent

On the convergence of gradient descent

Theorem | Smooth convex functions

Let f be a convex L -smooth scalar function over \mathbb{R}^n , and let $\alpha = L^{-1}$ the learning rate, then $\forall t \in [1, T]$:

$$f(\vec{x}^{(t)}) - f(\vec{x}^*) \leq \frac{2L}{T-1} \left\| \vec{x}^{(0)} - \vec{x}^* \right\|_2.$$

(see Gower (Télécom Paris))

From the last equation

$$f(\vec{x}^{(t+1)}) - f(\vec{x}^*) \leq f(\vec{x}^{(t)}) - f(\vec{x}^*) - \frac{1}{2L} \left\| \vec{\nabla} f(\vec{x}^{(t)}) \right\|_2,$$

apply the **convexity** property

$$f(\vec{x}^{(t)}) - f(\vec{x}^*) \leq \vec{\nabla} f(\vec{x}^{(t)}) \cdot (\vec{x}^{(t)} - \vec{x}^*) \leq \left\| \vec{\nabla} f(\vec{x}^{(t)}) \right\|_2^2 \left\| \vec{x}^{(t)} - \vec{x}^* \right\|_2^2$$

and reconstruct:

$$\Delta^{(t+1)} \leq \Delta^{(t)} - \beta \left(\Delta^{(t)} \right)^2 \Leftrightarrow \beta \leq \beta \frac{\Delta^{(t)}}{\Delta^{(t+1)}} \leq \frac{1}{\Delta^{(t+1)}} - \frac{1}{\Delta^{(t)}},$$

where $\Delta^{(t)} = f(\vec{x}^{(t)}) - f(\vec{x}^*)$ and

$$\beta = \frac{1}{2L \left\| \vec{x}^{(0)} - \vec{x}^* \right\|_2^2}, \quad \text{since} \quad \left\| \vec{x}^{(t)} - \vec{x}^* \right\|_2^2 \leq \left\| \vec{x}^{(0)} - \vec{x}^* \right\|_2^2.$$



Gradient Descent

On the convergence of gradient descent

Theorem | Smooth convex functions

Let f be a convex L -smooth scalar function over \mathbb{R}^n , and let $\alpha = L^{-1}$ the learning rate, then $\forall t \in [1, T]$:

$$f(\vec{x}^{(t)}) - f(\vec{x}^*) \leq \frac{2L}{T-1} \left\| \vec{x}^{(0)} - \vec{x}^* \right\|_2.$$

(see Gower (Télécom Paris))

We finally conclude by summing over all $t \in [1, T]$:

$$(T-1) \beta \leq \frac{1}{\Delta(T)} - \frac{1}{\Delta(1)} \leq \frac{1}{\Delta(T)} \leq \frac{1}{\Delta(t)}.$$

□



Loss Functions vs Metrics

Evaluating algorithms

Is there a difference between **metrics** and **loss functions**? Can I choose arbitrarily?



Loss Functions vs Metrics

Evaluating algorithms

Regression task

N.B. we are not discussing "good vs bad" metric/loss

Let $f_{\{\Theta, \Omega\}} : \mathbb{R}^n \rightarrow \mathbb{R}$, such that $\vec{x} \xrightarrow{f} \hat{y}$, a **regression** model

$$\begin{array}{ll} \text{evaluation} & \rightarrow \|y - \hat{y}\|_p \\ \text{loss} & \rightarrow \|y - \hat{y}\|_{p>0} \end{array}$$

A priori, we could use the loss as evaluation metric as well.



Loss Functions vs Metrics

Evaluating algorithms

Regression task

N.B. we are not discussing "good vs bad" metric/loss

Let $f_{\{\Theta, \Omega\}} : \mathbb{R}^n \rightarrow \mathbb{R}$, such that $\vec{x} \xrightarrow{f} \hat{y}$, a **regression** model

$$\begin{array}{ll} \text{evaluation} & \rightarrow \|y - \hat{y}\|_p \\ \text{loss} & \rightarrow \|y - \hat{y}\|_{p>0} \end{array}$$

A priori, we could use the loss as evaluation metric as well.

Classification task

N.B. we are not discussing "good vs bad" metric/loss

Let $f_{\{\Theta, \Omega\}} : \mathbb{R}^n \rightarrow \mathbb{R}$, such that $\vec{x} \xrightarrow{f} \hat{y}$ (probability of being positive sample), a **classification** model

$$\begin{array}{ll} \text{evaluation} & \rightarrow \text{accuracy} \\ \text{loss} & \rightarrow \mathbb{E}_C [\ln \hat{C}] \text{ (cross entropy)} \end{array}$$

Depending on the task, we might need a *continuous* loss, but not necessarily a continuous evaluation!



2. The ML Mindset

• Regularisation

Table of contents

1. Some History and Philosophy to Start

2. The ML Mindset

Data preparation

Validation and test sets

The variance vs bias trade-off

Learning objectives and loss functions

Regularisation

3. ML Algorithms

4. Neural Networks

5. Conclusions





Regularisation Techniques

Containing the overfit

Q: Is there a way to *actively* reduce overfitting the training data?



Regularisation Techniques

Containing the overfit

Q: Is there a way to *actively* reduce overfitting the training data?

regularised model = model + constraint on parameters



Regularisation Techniques

Containing the overfit

Q: Is there a way to *actively* reduce overfitting the training data?

$$\mathcal{L}_{\text{full}}(\Theta; \Omega) = \mathcal{L}(\Theta; \Omega) +$$



Regularisation Techniques

Containing the overfit

Q: Is there a way to *actively* reduce overfitting the training data?

$$\mathcal{L}_{\text{full}}(\Theta; \Omega, \Lambda) = \mathcal{L}(\Theta; \Omega) + \mathcal{L}_{\text{reg}}(\Theta; \Lambda)$$



Regularisation Techniques

Containing the overfit

Q: Is there a way to *actively* reduce overfitting the training data?

$$\mathcal{L}_{\text{full}}(\Theta; \Omega, \Lambda) = \mathcal{L}(\Theta; \Omega) + \mathcal{L}_{\text{reg}}(\Theta; \Lambda)$$

L_p regularisation

Define the L_p norm of $\vec{x} \in \mathbb{R}^k$:

$$\|\vec{x}\|_p = \left(\sum_{i=1}^p |x_i|^p \right)^{\frac{1}{p}}, \quad \text{special cases} \quad \begin{cases} L_0 & : \|\vec{x}\|_0 = \sum_{i=1}^p \delta_{|x_i|, 0} \\ L_\infty & : \|\vec{x}\|_\infty = \sup_{i \in [1, p]} |x_i| \end{cases}$$



Regularisation Techniques

Containing the overfit

Q: Is there a way to *actively* reduce overfitting the training data?

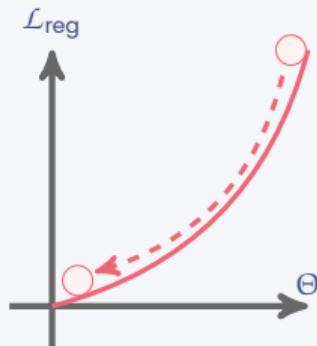
$$\mathcal{L}_{\text{full}}(\Theta; \Omega, \Lambda) = \mathcal{L}(\Theta; \Omega) + \mathcal{L}_{\text{reg}}(\Theta; \Lambda)$$

L_p regularisation

Then:

$$\mathcal{L}_{\text{reg}}(\Theta; \Lambda) = \lambda_p \|\Theta\|_p^p, \quad \text{s.t. } \lambda_p \in \Lambda.$$

Most common regularisation techniques are $p = 1$ (**LASSO**) and $p = 2$ (**Ridge**).





Regularisation Techniques

Containing the overfit

Q: Is there a way to *actively* reduce overfitting the training data?

$$\mathcal{L}_{\text{full}}(\Theta; \Omega, \Lambda) = \mathcal{L}(\Theta; \Omega) + \mathcal{L}_{\text{reg}}(\Theta; \Lambda)$$

L_1 and L_2 regularisation: probabilistic interpretation or simple trick?

Remember $\mathcal{P}(A \cap B) = \mathcal{P}(A | B)\mathcal{P}(B) = \mathcal{P}(B | A)\mathcal{P}(A)$ which implies (Bayes' theorem):

$$\underbrace{\mathcal{P}(A | B)}_{\text{posterior}} = \frac{\overbrace{\mathcal{P}(B | A)}^{\text{likelihood}} \overbrace{\mathcal{P}(A)}^{\text{prior}}}{\underbrace{\mathcal{P}(B)}_{\text{marginal}}}.$$



Regularisation Techniques

Containing the overfit

Q: Is there a way to *actively* reduce overfitting the training data?

$$\mathcal{L}_{\text{full}}(\Theta; \Omega, \Lambda) = \mathcal{L}(\Theta; \Omega) + \mathcal{L}_{\text{reg}}(\Theta; \Lambda)$$

L_1 and L_2 regularisation: probabilistic interpretation or simple trick?

$$\Theta^* = \arg \max_{\Theta} \left(\underbrace{\ln \mathcal{P}(\Theta | \vec{x})}_{\text{MAP}} \right) = \arg \max_{\Theta} \left(\underbrace{\ln \mathcal{P}(\vec{x} | \Theta)}_{\text{MLE}} + \underbrace{\ln \mathcal{P}(\Theta)}_{\text{prior}} \right)$$

where, for $\Theta = (\theta_1, \theta_2, \dots, \theta_p)$:

$$L_1: \quad \mathcal{P}(\Theta) = \text{Laplace}(\Theta | 0, b) = \frac{1}{2^p b^p} e^{-\frac{\|\Theta\|_1}{b}}, \quad L_2: \quad \mathcal{P}(\Theta) = \mathcal{N}(\Theta | 0, \sigma^2) = \frac{1}{(2\pi\sigma^2)^p} e^{-\frac{\|\Theta\|_2^2}{2\sigma^2}}.$$

N.B.: certainly “MAP w/ prior \Rightarrow penalised least squares”, but “penalised least squares $\not\Rightarrow$ Gaussian/Laplace prior”: it is rather a matter of efficiency and good results.

(see Gribonval (2011))



Regularisation Techniques

Containing the overfit

Q: Is there a way to *actively* reduce overfitting the training data?

$$\mathcal{L}_{\text{full}}(\Theta; \Omega, \Lambda) = \mathcal{L}(\Theta; \Omega) + \mathcal{L}_{\text{reg}}(\Theta; \Lambda)$$

L_1 and L_2 regularisation: probabilistic interpretation or simple trick?

$$\Theta^* = \arg \max_{\Theta} \left(\underbrace{\ln \mathcal{P}(\Theta | \vec{x})}_{\text{MAP}} \right) = \arg \max_{\Theta} \left(\underbrace{\ln \mathcal{P}(\vec{x} | \Theta)}_{\text{MLE}} + \underbrace{\ln \mathcal{P}(\Theta)}_{\text{prior}} \right)$$

where, for $\Theta = (\theta_1, \theta_2, \dots, \theta_p)$:

$$L_1: \quad \mathcal{P}(\Theta) = \text{Laplace}(\Theta | 0, b) = \frac{1}{2^p b^p} e^{-\frac{\|\Theta\|_1}{b}}, \quad L_2: \quad \mathcal{P}(\Theta) = \mathcal{N}(\Theta | 0, \sigma^2) = \frac{1}{(2\pi\sigma^2)^p} e^{-\frac{\|\Theta\|_2^2}{2\sigma^2}}.$$

Q: \mathcal{L}_{reg} adds restrictions on the parameters of the model to contain the overfit. How to interpret this in terms of **bias vs variance**?

(see Gribonval (2011))



Regularisation Techniques

Containing the overfit

Q: Is there a way to *actively* reduce overfitting the training data?

$$\mathcal{L}_{\text{full}}(\Theta; \Omega, \Lambda) = \mathcal{L}(\Theta; \Omega) + \mathcal{L}_{\text{reg}}(\Theta; \Lambda)$$

L_1 and L_2 regularisation: probabilistic interpretation or simple trick?

$$\Theta^* = \arg \max_{\Theta} \left(\underbrace{\ln \mathcal{P}(\Theta | \vec{x})}_{\text{MAP}} \right) = \arg \max_{\Theta} \left(\underbrace{\ln \mathcal{P}(\vec{x} | \Theta)}_{\text{MLE}} + \underbrace{\ln \mathcal{P}(\Theta)}_{\text{prior}} \right)$$

where, for $\Theta = (\theta_1, \theta_2, \dots, \theta_p)$:

$$L_1: \quad \mathcal{P}(\Theta) = \text{Laplace}(\Theta | 0, b) = \frac{1}{2^p b^p} e^{-\frac{\|\Theta\|_1}{b}}, \quad L_2: \quad \mathcal{P}(\Theta) = \mathcal{N}(\Theta | 0, \sigma^2) = \frac{1}{(2\pi\sigma^2)^p} e^{-\frac{\|\Theta\|_2^2}{2\sigma^2}}.$$

Q: \mathcal{L}_{reg} adds restrictions on the parameters of the model to contain the overfit. How to interpret this in terms of **bias vs variance**? Increase in bias, decrease in variance.

(see Gribonval (2011))



3. ML Algorithms

Taxonomy of algorithms

Table of contents

1. Some History and Philosophy to Start

2. The ML Mindset

3. ML Algorithms

Taxonomy of algorithms

Unsupervised learning

Supervised learning

Ensemble learning

4. Neural Networks

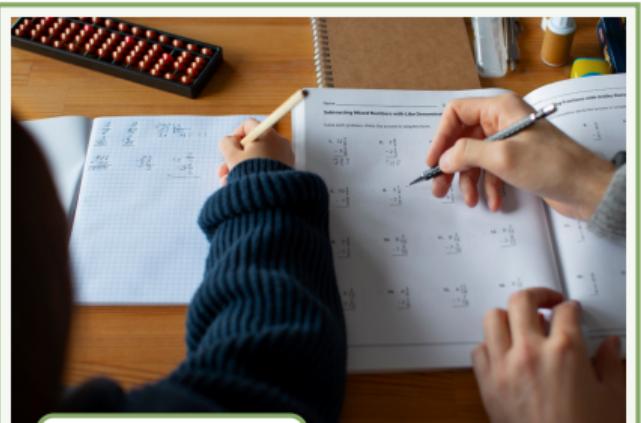
5. Conclusions





Types of Algorithms

A simple distinction



SUPERVISED

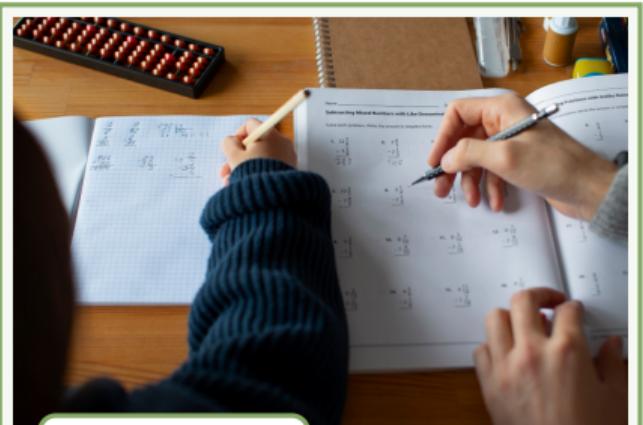


UNSUPERVISED



Types of Algorithms

A simple distinction



SUPERVISED



UNSUPERVISED

Supervised learning

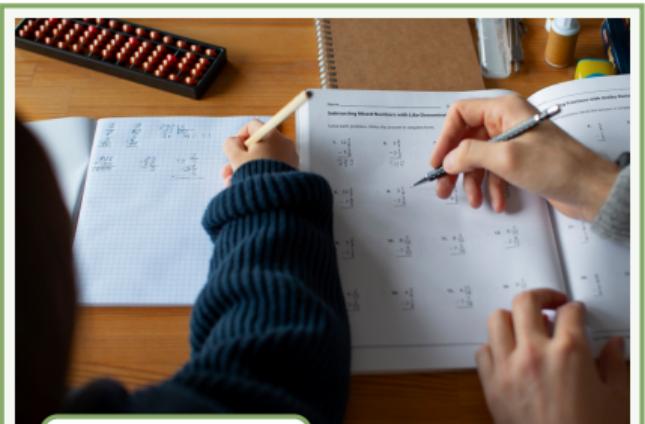
- “learn” knowing the **result** (labels)
- **iterative** process with examples
- need **annotated** data





Types of Algorithms

A simple distinction



SUPERVISED



UNSUPERVISED

Supervised learning

- “learn” knowing the **result** (labels)
- **iterative** process with examples
- need **annotated** data

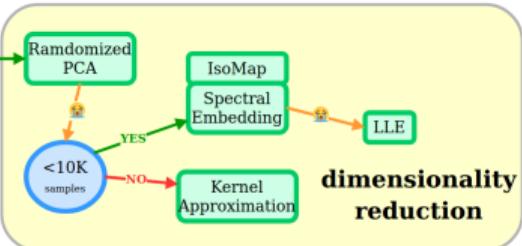
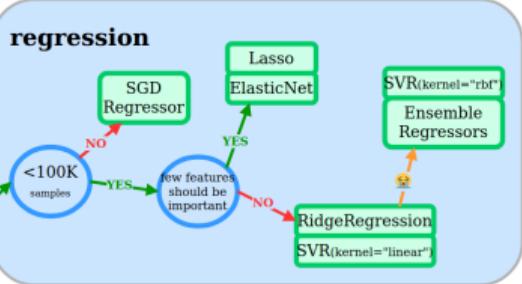
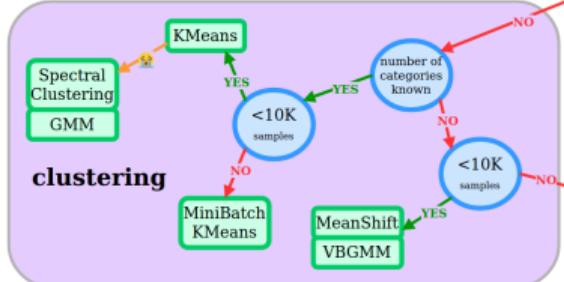
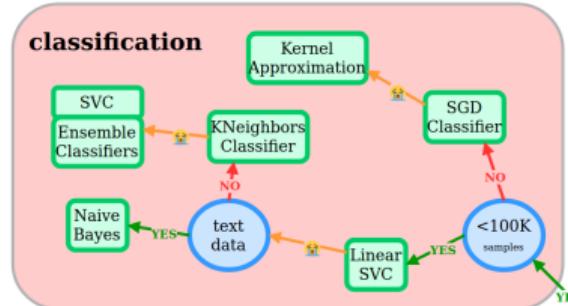
Unsupervised learning

- “learn” a **structure** in the data
- can identify usable **patterns**
- data are **not labelled**



Types of Algorithms

A more realistic scenario



scikit-learn
algorithm cheat
sheet



Types of Algorithms

Some more details...

Supervised learning

Let $\mathcal{D} = \{\vec{x}, y\}$ a **labelled** dataset:

$$f_{\text{supervised}} : \mathbb{K}^p \rightarrow \mathbb{K}$$

- regression
- classification
- time series inference
- (LLMs, generative AI, etc. debatable)

Unsupervised learning

Let $\mathcal{D} = \{\vec{x}\}$ a set of **data points**:

$$f_{\text{unsupervised}} : \mathbb{K}^p \rightarrow \mathbb{K}^q$$

- principal components analysis
- clustering and manifold learning
- anomaly detection
- etc.



Types of Algorithms

A wide spectrum

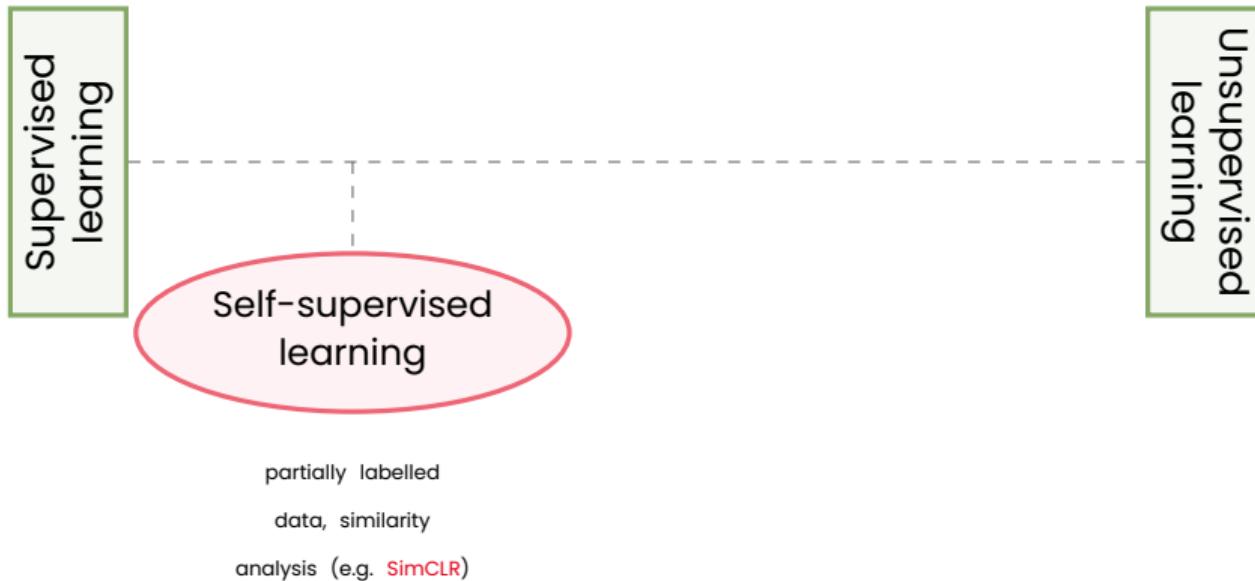
Supervised learning

Unsupervised learning



Types of Algorithms

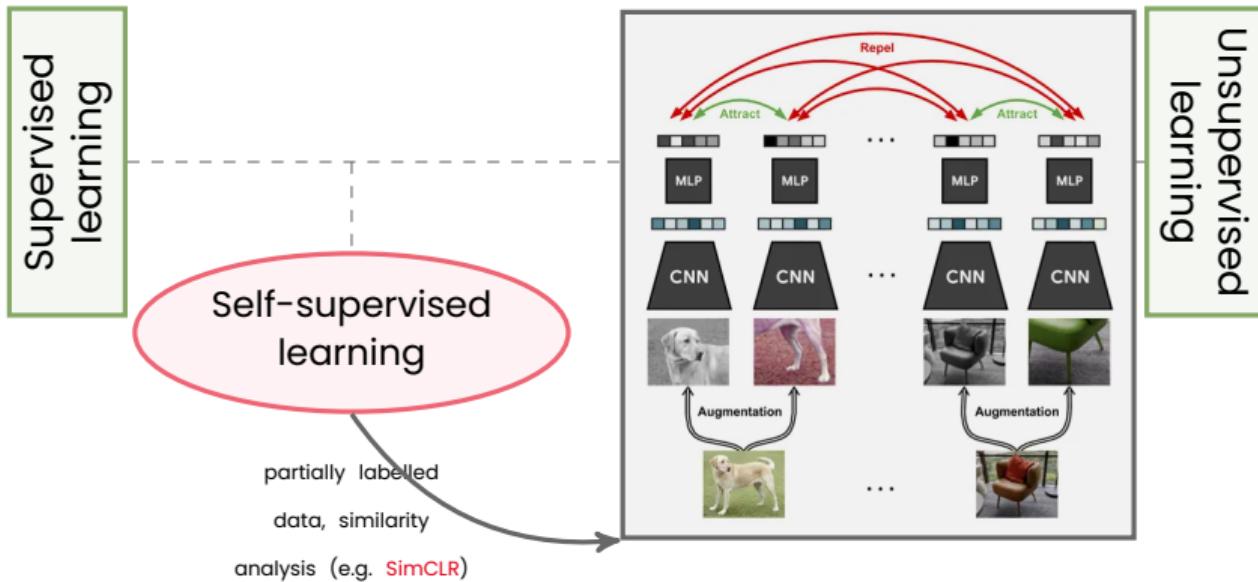
A wide spectrum





Types of Algorithms

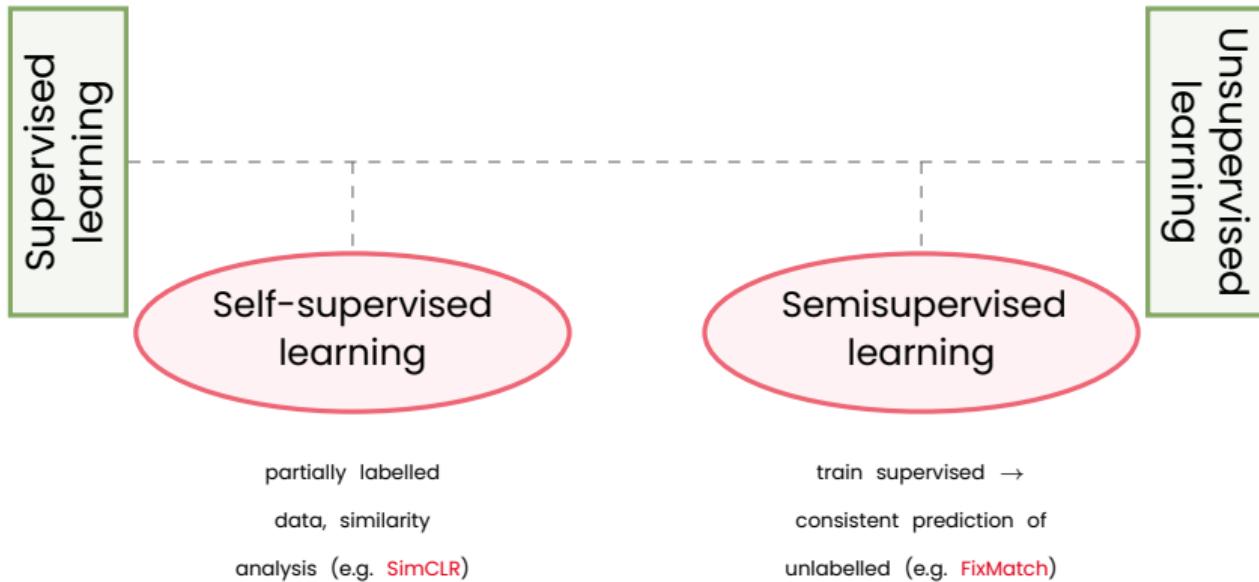
A wide spectrum





Types of Algorithms

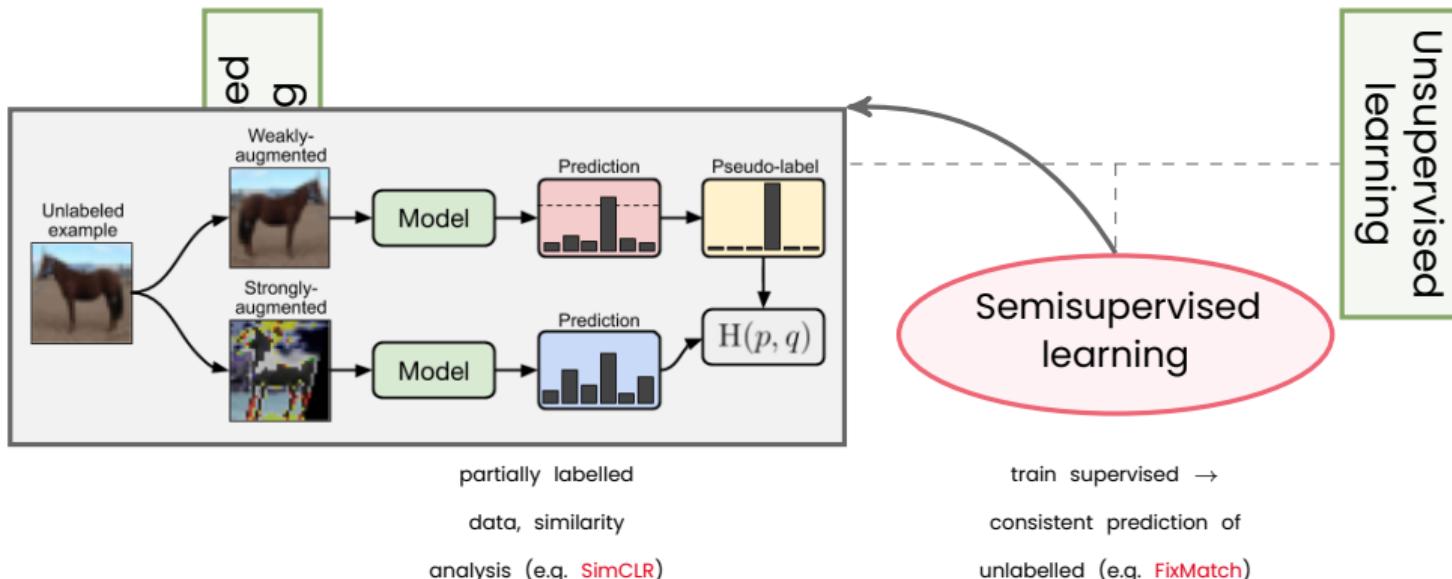
A wide spectrum





Types of Algorithms

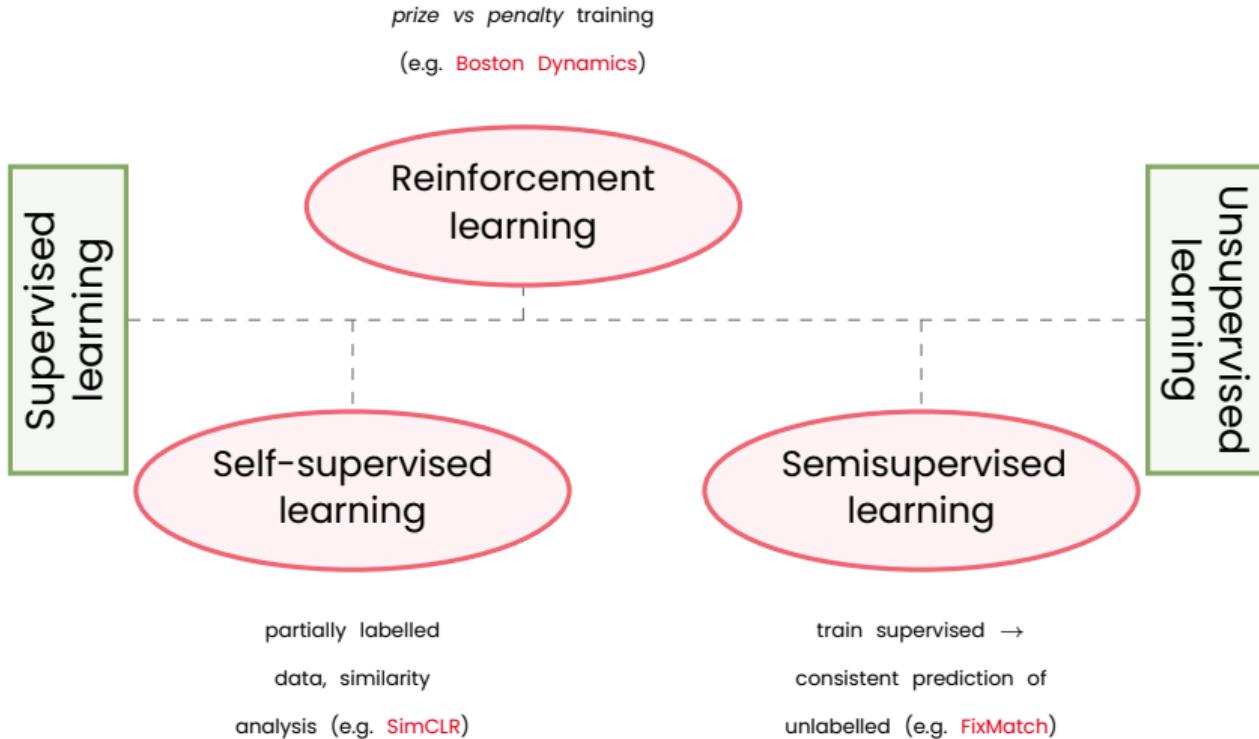
A wide spectrum





Types of Algorithms

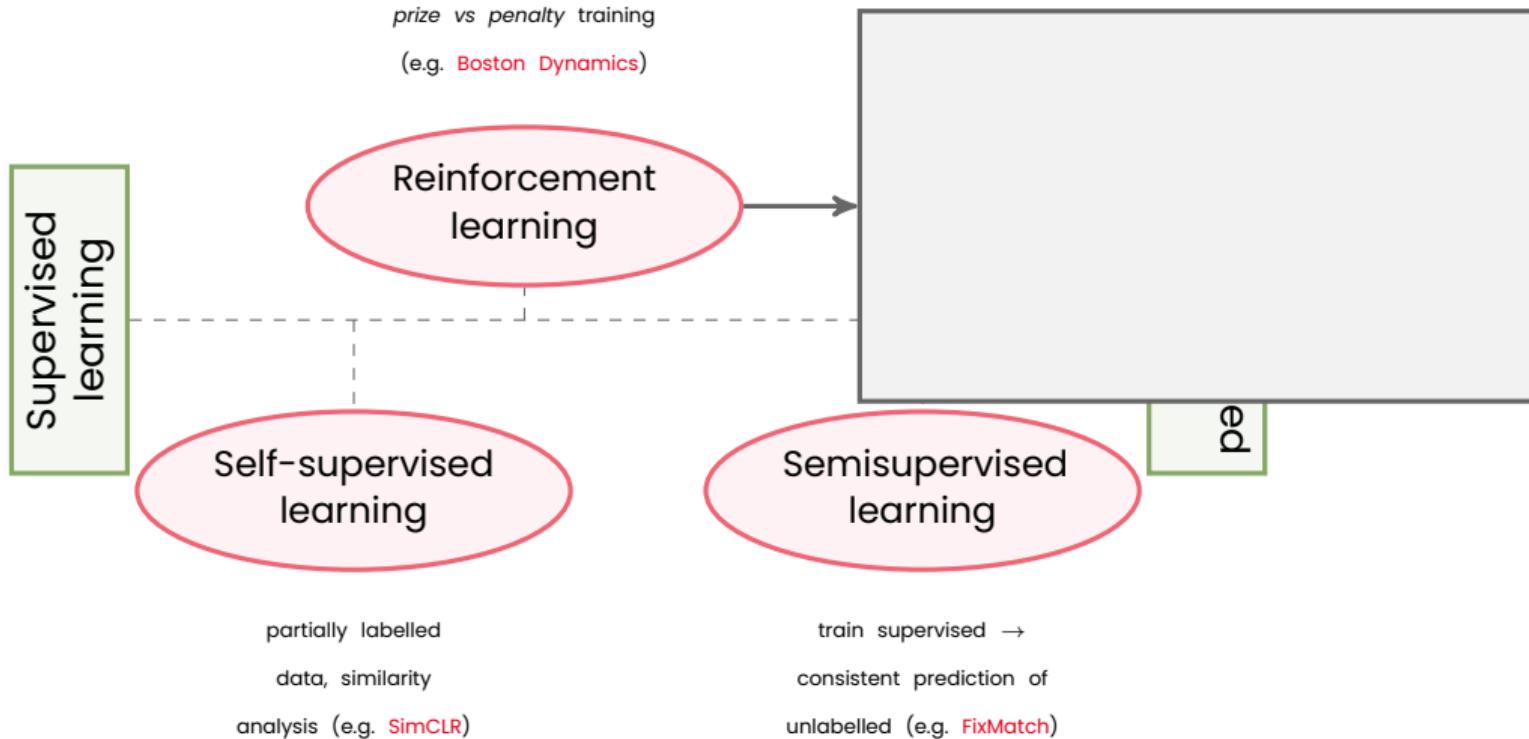
A wide spectrum





Types of Algorithms

A wide spectrum





3. ML Algorithms

Unsupervised learning

Table of contents

1. Some History and Philosophy to Start

2. The ML Mindset

3. ML Algorithms

Taxonomy of algorithms

Unsupervised learning

Supervised learning

Ensemble learning

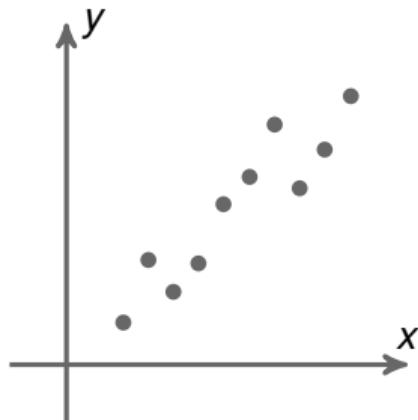
4. Neural Networks

5. Conclusions



Principal Components Analysis

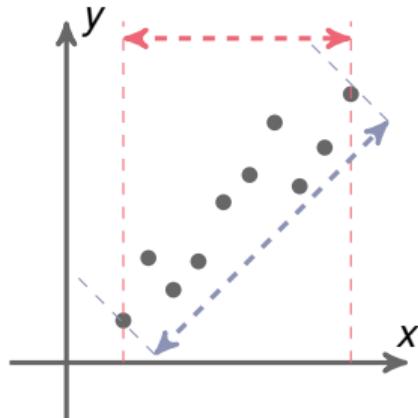
Definition





Principal Components Analysis

Definition

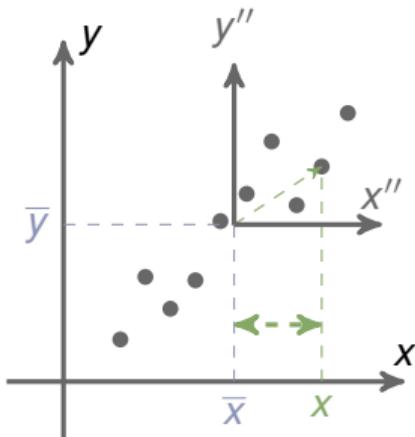


- data better from another angle ($\leftarrow \rightarrow > \leftarrow \rightarrow$)



Principal Components Analysis

Definition

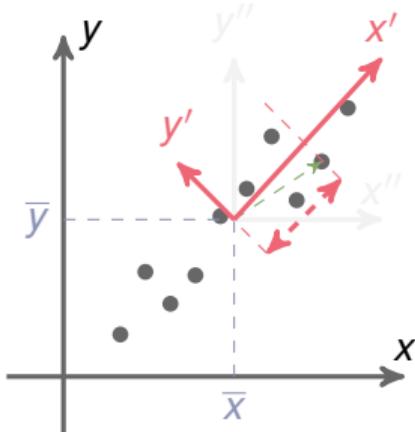


- data better from another angle ($\longleftrightarrow > \longleftrightarrow$)
- “distance” from centre



Principal Components Analysis

Definition

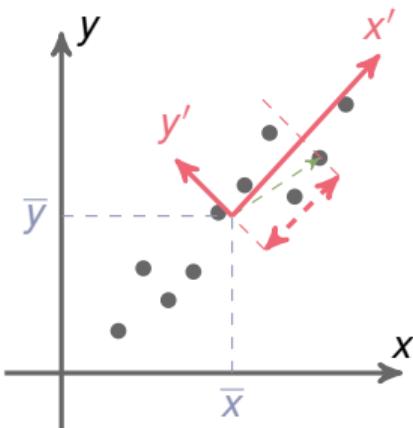


- data better from another angle (\angle)
- “distance” from centre
- find maximal “distance”²



Principal Components Analysis

Definition

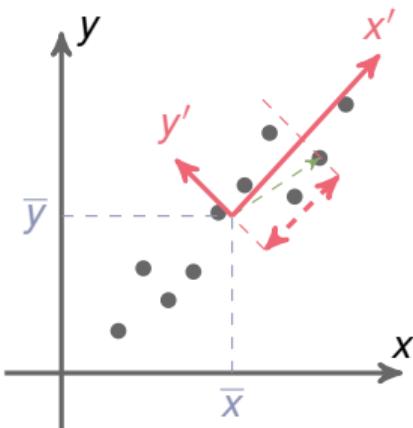


- data better from another angle ($\angle > \angle$)
- “distance” from centre
- find maximal “distance²” (i.e. maximal **variance**)



Principal Components Analysis

Definition



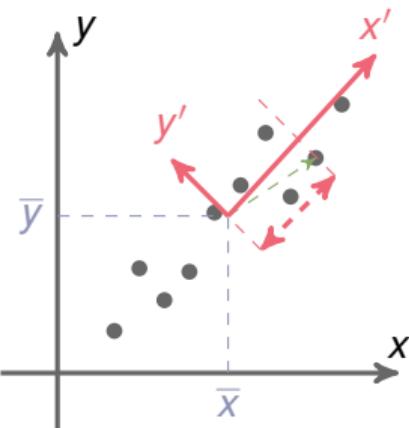
Let $\vec{x}_i \in \mathbb{R}^p$ for $i = 1, 2, \dots, n$ s.t. $E[\vec{x}] = \bar{\vec{x}}$. Call $\vec{y}_i = \vec{x}_i - \bar{\vec{x}}$ ($i = 1, 2, \dots, n$) the **centred** data.

- data better from another angle ($\leftrightarrow > \leftrightarrow$)
- “distance” from centre
- find maximal “distance²” (i.e. maximal variance)



Principal Components Analysis

Definition



- data better from another angle ($\leftrightarrow > \leftrightarrow$)
- “distance” from centre
- find maximal “distance²” (i.e. maximal variance)

Let $\vec{x}_i \in \mathbb{R}^p$ for $i = 1, 2, \dots, n$ s.t. $E[\vec{x}] = \bar{\vec{x}}$. Call $\vec{y}_i = \vec{x}_i - \bar{\vec{x}}$ ($i = 1, 2, \dots, n$) the **centred** data.

Preliminaries (spectral theorem)

Let $M \in \mathbb{R}^{p \times p}$ s.t. $M^T = M$. Then, \exists complete orthonormal basis of $\mathbb{R}^p \{ \vec{e}_1, \vec{e}_2, \dots, \vec{e}_p \}$ s.t.

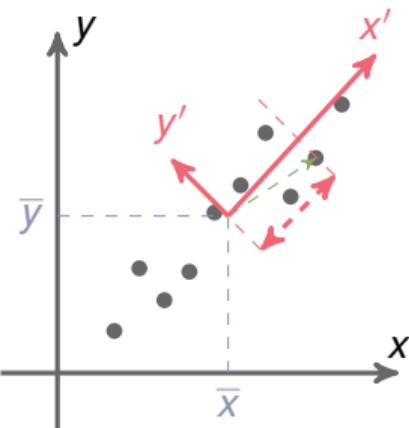
$$M = \left(\vec{m}_i^T \right)_{i=1,2,\dots,p} = \sum_{i=1}^p \lambda_i \vec{e}_i \vec{e}_i^T,$$

where $\lambda_i \in \mathbb{R}^+ \quad \forall i \in [1, p]$.



Principal Components Analysis

Definition



- data better from another angle ($\longleftrightarrow > \longleftrightarrow$)
- “distance” from centre
- find maximal “distance²” (i.e. maximal variance)

Let $\vec{x}_i \in \mathbb{R}^p$ for $i = 1, 2, \dots, n$ s.t. $E[\vec{x}] = \bar{\vec{x}}$. Call $\vec{y}_i = \vec{x}_i - \bar{\vec{x}}$ ($i = 1, 2, \dots, n$) the **centred** data.

Maximal variance

Let $Y \in \mathbb{R}^{n \times p}$ matrix representation of data with **covariance** $C = n^{-1}X^T X$.

We look for a *new basis*

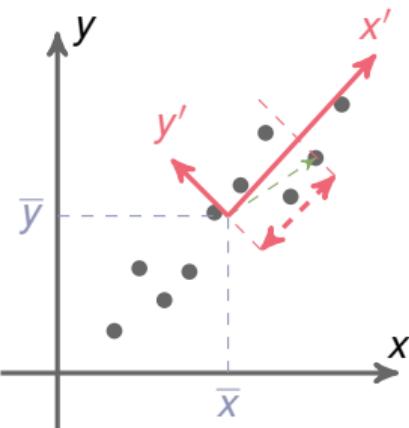
$$\underbrace{Y'}_{\text{scores / principal components}} = Y \underbrace{W}_{\text{loadings}}$$

which maximises the variance in each direction of the vectors $\vec{y}'_{[i]} \in \mathbb{R}^p$ ($i = 1, 2, \dots, n$).



Principal Components Analysis

Definition



- data better from another angle ($\longleftrightarrow > \longleftrightarrow$)
- “distance” from centre
- find maximal “distance²” (i.e. maximal variance)

Let $\vec{x}_i \in \mathbb{R}^p$ for $i = 1, 2, \dots, n$ s.t. $E[\vec{x}] = \bar{\vec{x}}$. Call $\vec{y}_i = \vec{x}_i - \bar{\vec{x}}$ ($i = 1, 2, \dots, n$) the **centred** data.

Maximal variance

We need:

$$\text{Var}(y'_{[i](a)}) = \text{Var}(\vec{y}_{[i]} \cdot \vec{w}_{(a)}) = \vec{w}_{(a)}^T C_{[i]} \vec{w}_{(a)}.$$

and compute

$$\arg \max_{\vec{w}_{(a)}} \vec{w}_{(a)}^T C_{[i]} \vec{w}_{(a)}$$

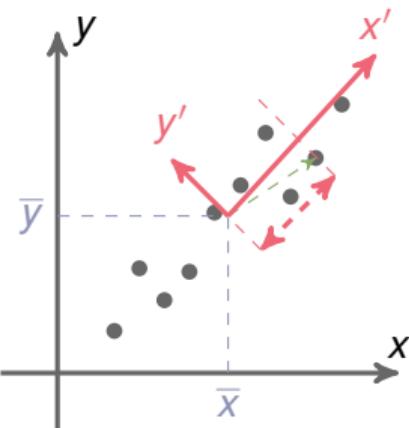
constrained to

$$\vec{w}_{(a)} \cdot \vec{w}_{(b)} = \delta_{ab}.$$



Principal Components Analysis

Definition



- data better from another angle ($\leftrightarrow > \leftrightarrow$)
- “distance” from centre
- find maximal “distance²” (i.e. maximal variance)

Let $\vec{x}_i \in \mathbb{R}^p$ for $i = 1, 2, \dots, n$ s.t. $E[\vec{x}] = \bar{\vec{x}}$. Call $\vec{y}_i = \vec{x}_i - \bar{\vec{x}}$ ($i = 1, 2, \dots, n$) the **centred** data.

Principal Components (theorem)

If C has distinct eigenvalues $\lambda_1, \dots, \lambda_p$, then

$\vec{w}_{(a)}$ is the eigenvector corresponding to the a -th largest eigenvector λ_a .

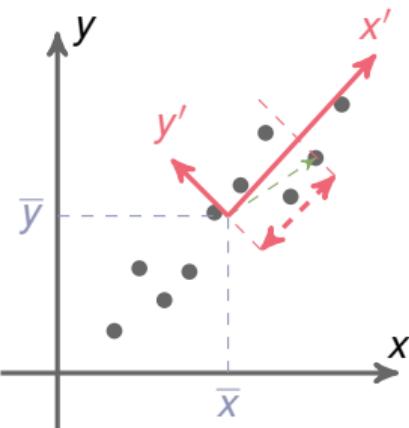
Moreover,

$$\text{Var}(y'_{[i](a)}) = \vec{w}_{(a)}^T C_{[i]} \vec{w}_{(a)} = \lambda_a.$$



Principal Components Analysis

Definition



- data better from another angle ($\leftrightarrow > \leftrightarrow$)
- “distance” from centre
- find maximal “distance²” (i.e. maximal variance)

Let $\vec{x}_i \in \mathbb{R}^p$ for $i = 1, 2, \dots, n$ s.t. $E[\vec{x}] = \bar{\vec{x}}$. Call $\vec{y}_i = \vec{x}_i - \bar{\vec{x}}$ ($i = 1, 2, \dots, n$) the **centred** data.

Principal Components (theorem)

If C has distinct eigenvalues $\lambda_1, \dots, \lambda_p$, then

$\vec{w}_{(a)}$ is the eigenvector corresponding to the a -th largest eigenvector λ_a .

Moreover,

$$\text{Var}(y'_{[i](a)}) = \vec{w}_{(a)}^T C_{[i]} \vec{w}_{(a)} = \lambda_a.$$

HOMEWORK

- Prove the PC theorem. Proceed iteratively from $\vec{w}_{(1)}$.



Principal Components Analysis

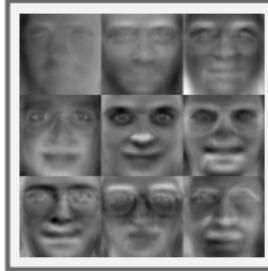
Dimensionality reduction | Eigenfaces

original images ([Olivetti dataset](#))



PCA (on image vectors)

eigenface basis





Principal Components Analysis

Dimensionality reduction | Eigenfaces

original images (Olivetti dataset)



PCA
(on image vectors)

eigenface basis



1. let $Z \in [0, 255]^2$ be a grayscale image
2. define $\vec{y} = \text{vec}(Z)$, then find *basis* of images $\{\vec{w}_{(1)}, \dots, \vec{w}_{(p)}\}$
3. write each image $\vec{y}_{[i]} = \sum_{k=1}^p y'_{[i](k)} \vec{w}_{(k)}$, where $y'_{[i](k)} = \vec{y}_{[i]} \cdot \vec{w}_{(k)}$
4. (optional) use $\vec{y}'_{[i]} \in \mathbb{R}^p$ to train a classifier (see Sirovich and Kirby (1987) and Turk and Petland (1991))



Principal Components Analysis

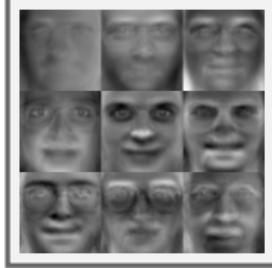
Dimensionality reduction | Eigenfaces

original images (Olivetti dataset)



PCA (on image vectors)

eigenface basis



As $\lambda_{(k)}$ represents the **variance explained** by the k -th principal component:

$$\vec{y} \quad W = \vec{y}'$$



Principal Components Analysis

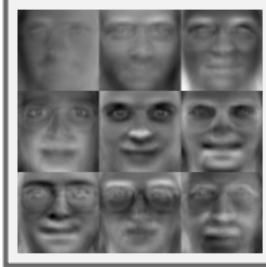
Dimensionality reduction | Eigenfaces

original images (Olivetti dataset)



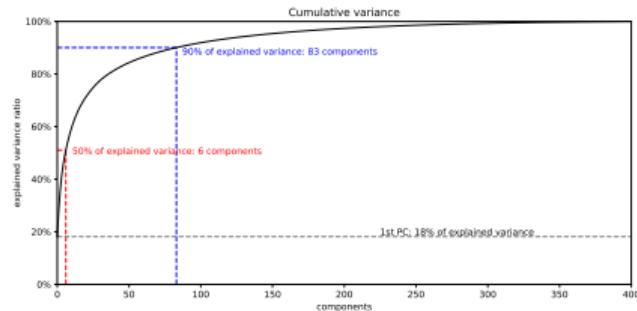
PCA (on image vectors)

eigenface basis



As $\lambda_{(k)}$ represents the **variance explained** by the k -th principal component:

$$\vec{y} \quad W_\alpha = \vec{y}'_\alpha$$

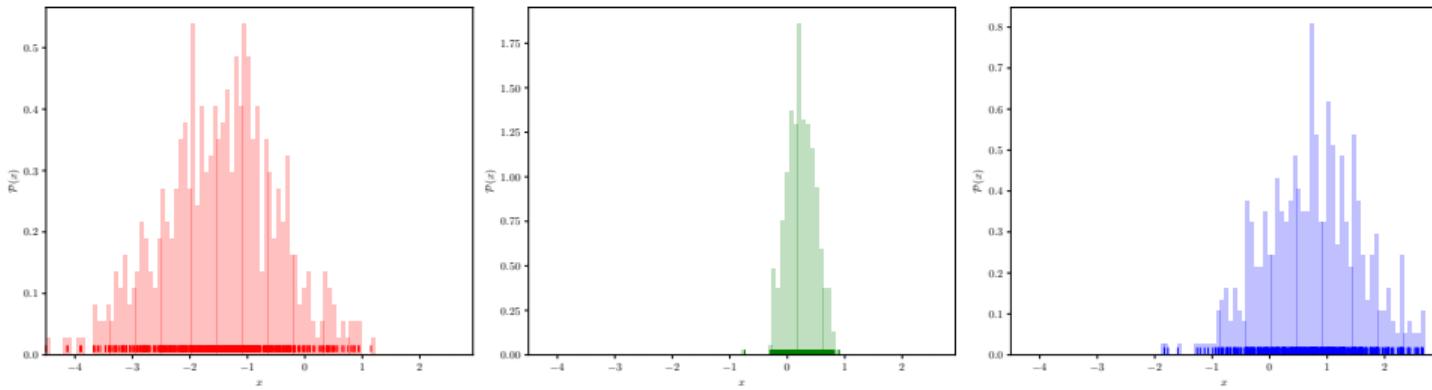




Gaussian Mixture Model

Definition

Let $\mathcal{D} = \{x_i \in \mathbb{R} \mid i = 1, 2, \dots, n\}$ s.t.



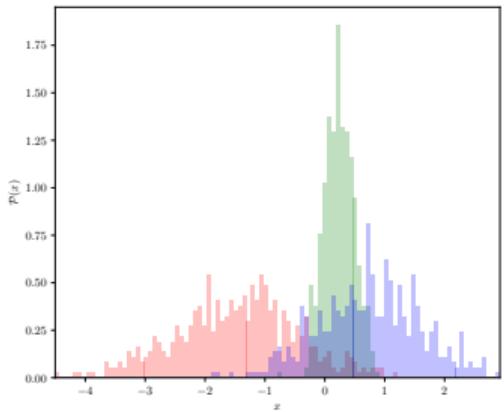
hypothesis: data are sampled from different Gaussian distributions

objective: can we group data according to the parameters of different Gaussian distributions?



Gaussian Mixture Model

Definition

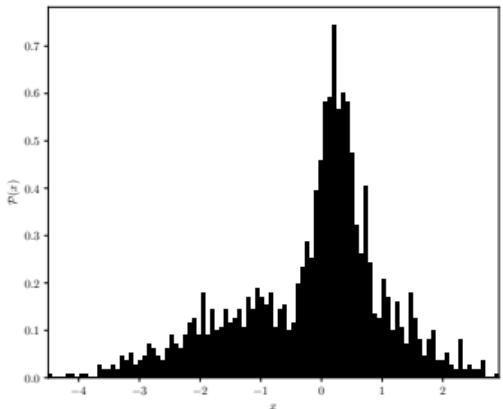


It looks easy knowing the ground truth...



Gaussian Mixture Model

Definition



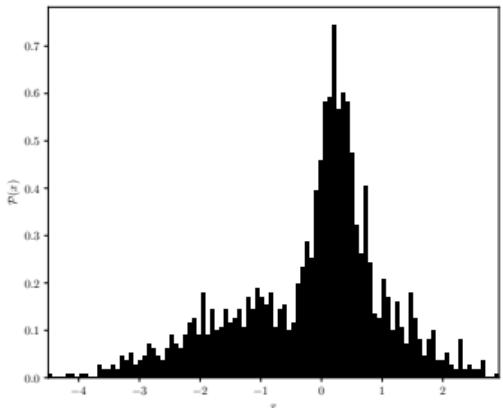
We need to build several normal distributions, then select what we need → build a *Gaussian Mixture Model*

...but it is not! (even 1D!)



Gaussian Mixture Model

Definition



...but it is not! (even 1D!)

We need to build several normal distributions, then select what we need → build a *Gaussian Mixture Model*

Reminders

Remember:

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

and (Bayes' theorem):

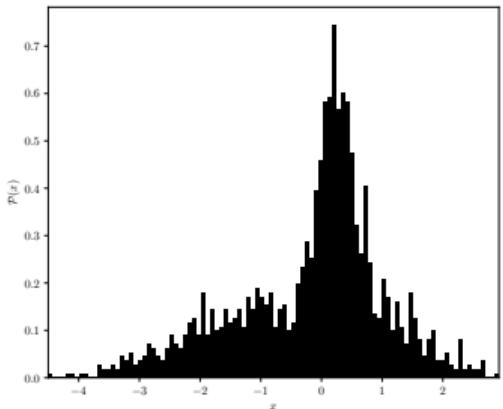
$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}.$$





Gaussian Mixture Model

Definition



Gaussian Mixture Model

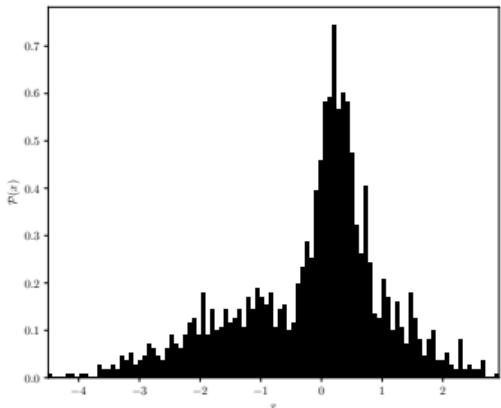
Consider $K > 1$ components and let
 $\Theta = \{(c_{(1)}, \mu_1, \sigma_1^2), \dots, (c_{(K)}, \mu_{(K)}, \sigma_{(K)}^2)\}$, where $c_{(k)}$
($k = 1, 2, \dots, K$) are the probabilities of picking
the k -th Gaussian.

...but it is not! (even 1D!)



Gaussian Mixture Model

Definition



...but it is not! (even 1D!)

Gaussian Mixture Model

Consider $K > 1$ components and let $\Theta = \{(c_{(1)}, \mu_1, \sigma_1^2), \dots, (c_{(K)}, \mu_{(K)}, \sigma_{(K)}^2)\}$, where $c_{(k)}$ ($k = 1, 2, \dots, K$) are the probabilities of picking the k -th Gaussian.

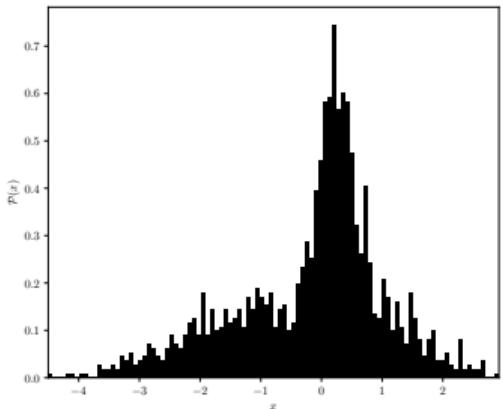
We observe the *marginal* (c is latent) likelihood:

$$\begin{aligned} \mathcal{P}(x | \Theta) &= \sum_{k=1}^K \mathcal{P}(C = k | \Theta) \mathcal{P}(x | C = k; \Theta) \\ &= \sum_{k=1}^K c_{(k)} \mathcal{N}(x | \mu_{(k)}, \sigma_{(k)}^2). \end{aligned}$$



Gaussian Mixture Model

Definition



$$\begin{aligned} \mathcal{P}(x \mid \Theta) &= \sum_{k=1}^K \mathcal{P}(C = k \mid \Theta) \mathcal{P}(x \mid C = k; \Theta) \\ &= \sum_{k=1}^K c_{(k)} \mathcal{N}(x \mid \mu_{(k)}, \sigma_{(k)}^2). \end{aligned}$$

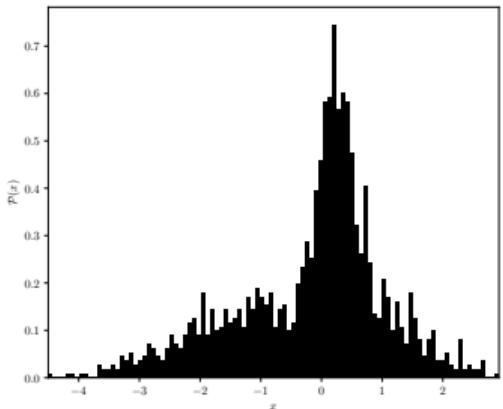
Gaussian Mixture Model

Consider $K > 1$ components and let $\Theta = \{(c_{(1)}, \mu_1, \sigma_1^2), \dots, (c_{(K)}, \mu_{(K)}, \sigma_{(K)}^2)\}$, where $c_{(k)}$ ($k = 1, 2, \dots, K$) are the probabilities of picking the k -th Gaussian.



Gaussian Mixture Model

Definition



$$\begin{aligned} p(x \mid \Theta) &= \sum_{k=1}^K p(C=k \mid \Theta) p(x \mid C=k; \Theta) \\ &= \sum_{k=1}^K c_{(k)} \mathcal{N}(x \mid \mu_{(k)}, \sigma_{(k)}^2). \end{aligned}$$

Gaussian Mixture Model

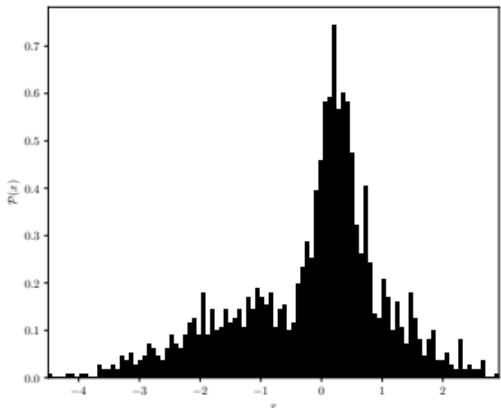
How to estimate the parameters? Ideally, we would like to assign any sample $x \in \mathcal{D}$ to a generating distribution:

$$\begin{aligned} \Theta^* &= \arg \max_{\Theta} \prod_{i=1}^n \mathcal{P}(x_i \mid \Theta) \\ &= \arg \max_{\Theta} \sum_{i=1}^n \ln \mathcal{P}(x_i \mid \Theta) \end{aligned}$$



Gaussian Mixture Model

Definition



$$\begin{aligned} p(x \mid \Theta) &= \sum_{k=1}^K p(C=k \mid \Theta) p(x \mid C=k; \Theta) \\ &= \sum_{k=1}^K c_{(k)} \mathcal{N}(x \mid \mu_{(k)}, \sigma_{(k)}^2). \end{aligned}$$

Gaussian Mixture Model

How to estimate the parameters? Ideally, we would like to assign any sample $x \in \mathcal{D}$ to a generating distribution:

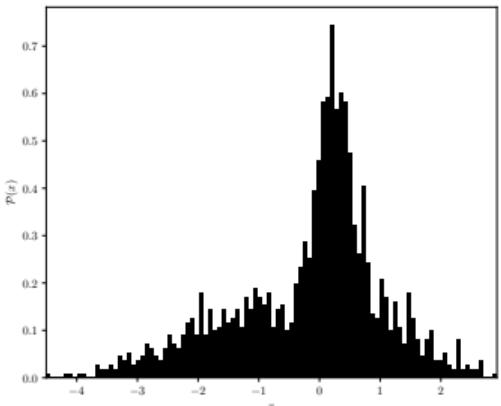
$$\begin{aligned} \Theta^* &= \arg \max_{\Theta} \prod_{i=1}^n \mathcal{P}(x_i \mid \Theta) \\ &= \arg \max_{\Theta} \sum_{i=1}^n \ln \mathcal{P}(x_i \mid \Theta) \end{aligned}$$

Use the **Expectation-Maximisation (EM)** algorithm. (EM is a technique to estimate the MLE of a latent variable model)



Gaussian Mixture Model

Definition



$$\begin{aligned} P(x \mid \Theta) &= \sum_{k=1}^K P(C = k \mid \Theta) P(x \mid C = k; \Theta) \\ &= \sum_{k=1}^K c_{(k)} \mathcal{N}(x \mid \mu_{(k)}, \sigma_{(k)}^2). \end{aligned}$$

Expectation Maximisation

Consider

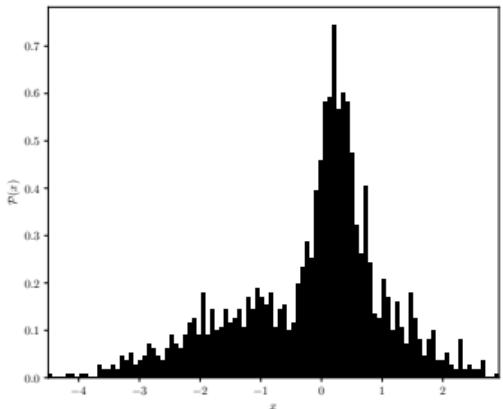
$$\begin{aligned} P(C = k \mid x_i; \Theta) &= \frac{P(x_i \mid C = k; \Theta) P(C = k)}{P(x_i \mid \Theta)} \\ &= \frac{P(x_i \mid C = k; \Theta) P(C = k)}{\sum_{k=1}^K P(x_i \mid C = k; \Theta) P(C = k)} \\ &= \frac{c_{(k)} \mathcal{N}(x_i \mid \mu_{(k)}, \sigma_{(k)}^2)}{\sum_{k=1}^K c_{(k)} \mathcal{N}(x_i \mid \mu_{(k)}, \sigma_{(k)}^2)} = \gamma_{i(k)} \end{aligned}$$

$$\text{N.B.: } \sum_{k=1}^K \gamma_{i(k)} = 1.$$



Gaussian Mixture Model

Definition



$$\begin{aligned} p(x \mid \Theta) &= \sum_{k=1}^K p(C=k \mid \Theta) p(x \mid C=k; \Theta) \\ &= \sum_{k=1}^K c_{(k)} \mathcal{N}(x \mid \mu_{(k)}, \sigma_{(k)}^2). \end{aligned}$$

Expectation Maximisation

$$\begin{aligned} Q(\Theta) &= \mathbb{E}_{P(C|x;\Theta)} [\ln P(x \mid \Theta)] \\ &= \sum_{i=1}^n \sum_{k=1}^K p(C=k \mid x_i; \Theta) \ln p(x_i \mid \Theta) \\ &= \sum_{i=1}^n \sum_{k=1}^K \gamma_{i(k)} \ln \left(c_{(k)} \mathcal{N}(x_i \mid \mu_{(k)}, \sigma_{(k)}^2) \right) \end{aligned}$$

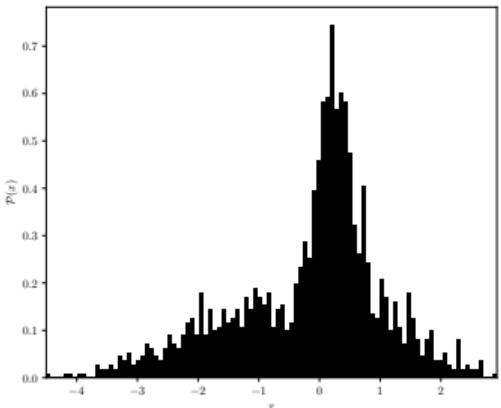
Compute (until convergence to $\Theta^{[t]} \xrightarrow{t \geq 1} \Theta^*$):

$$\Theta^{[t+1]} = \arg \max_{\Theta} Q(\Theta^{[t]})$$



Gaussian Mixture Model

Definition



$$\begin{aligned} p(x \mid \Theta) &= \sum_{k=1}^K p(C = k \mid \Theta) p(x \mid C = k; \Theta) \\ &= \sum_{k=1}^K c_{(k)} \mathcal{N}(x \mid \mu_{(k)}, \sigma_{(k)}^2). \end{aligned}$$

Expectation Maximisation

Compute (until convergence to $\Theta^{(t)} \xrightarrow{t \gg 1} \Theta^*$):

$$\Theta^{(t+1)} = \arg \max_{\Theta} Q(\Theta^{(t)}),$$

that is:

$$c_{(k)}^{(t+1)} = \mathbb{E}_{\mathcal{D}} [\gamma_{(k)}^{(t)}]$$

$$\mu_{(k)}^{(t+1)} = \frac{\mathbb{E}_{\mathcal{D}} [\gamma_{(k)}^{(t)} x]}{\mathbb{E}_{\mathcal{D}} [\gamma_{(k)}^{(t)}]}$$

$$\sigma_{(k)}^{(t+1)} = \frac{\mathbb{E}_{\mathcal{D}} [\gamma_{(k)}^{(t)} (x - \mu_{(k)})^2]}{\mathbb{E}_{\mathcal{D}} [\gamma_{(k)}^{(t)}]}$$

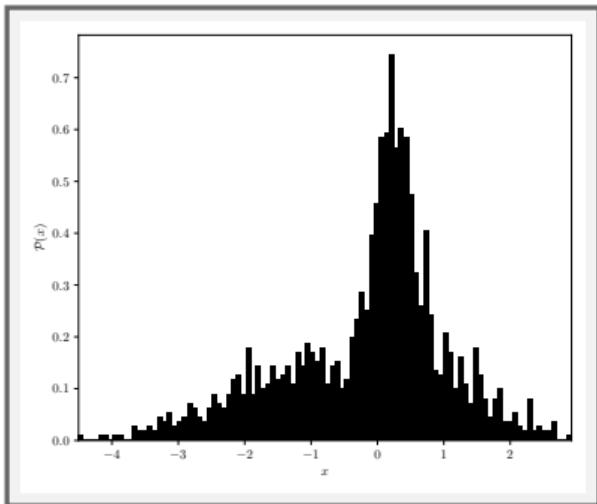


Gaussian Mixture Model

Unsupervised learning | Clustering

Remember

$$\gamma_{i(k)}^* = \mathcal{P}(C = k \mid x_i; \Theta^*) \Rightarrow \hat{C}_i = \arg \max_k \text{softmax}(\gamma_{i(k)}^*)$$



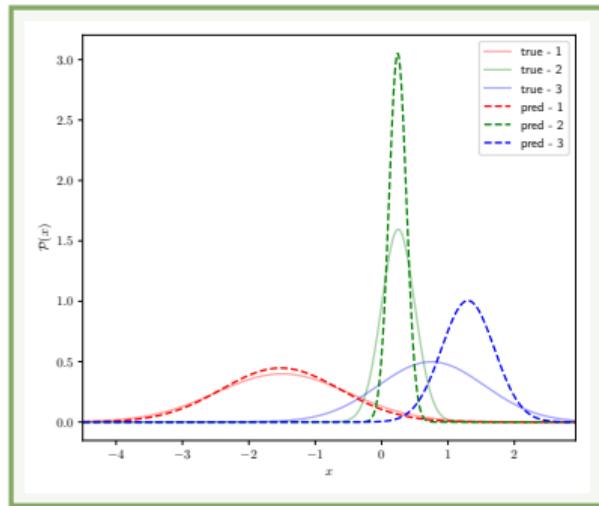
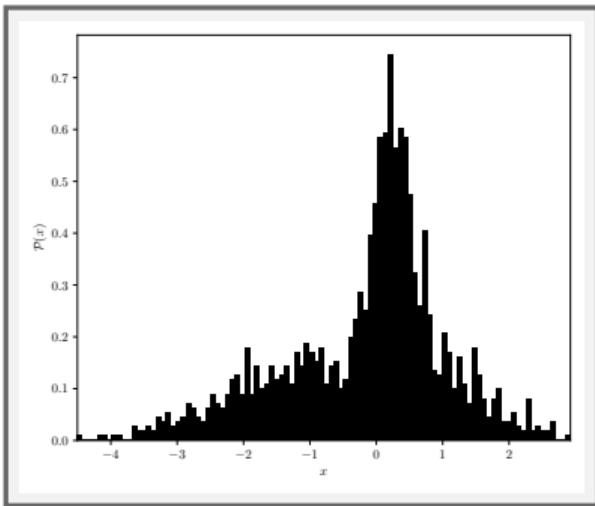


Gaussian Mixture Model

Unsupervised learning | Clustering

Remember

$$\gamma_{i(k)}^* = \mathcal{P}(C = k \mid x_i; \Theta^*) \Rightarrow \hat{C}_i = \arg \max_k \text{softmax}(\gamma_{i(k)}^*)$$



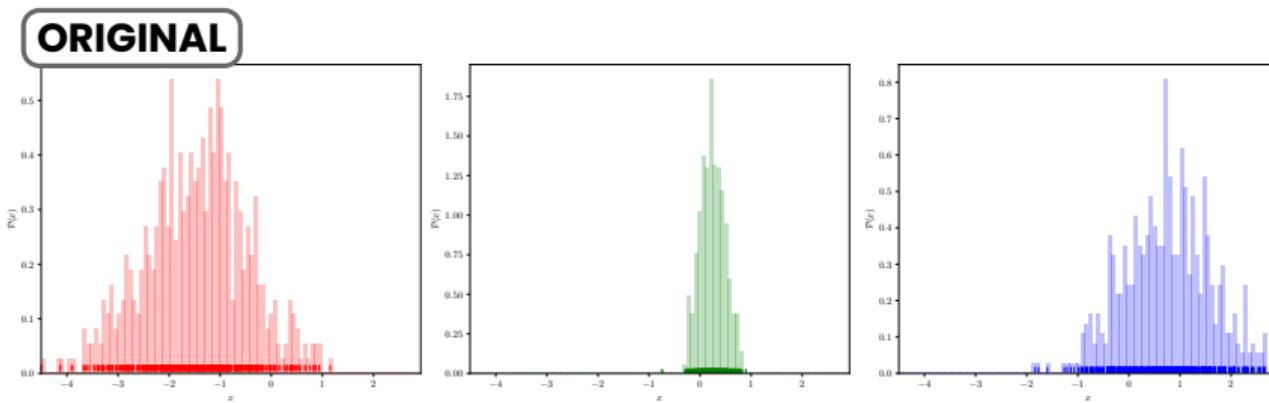


Gaussian Mixture Model

Unsupervised learning | Clustering

Remember

$$\gamma_{i(k)}^* = \mathcal{P}(C = k \mid x_i; \Theta^*) \Rightarrow \hat{C}_i = \arg \max_k \text{softmax}(\gamma_{i(k)}^*)$$



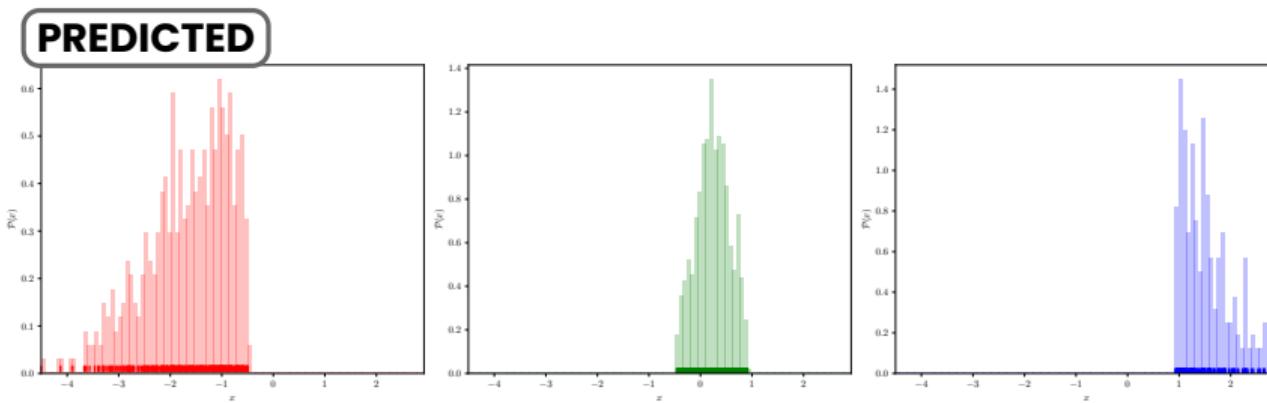


Gaussian Mixture Model

Unsupervised learning | Clustering

Remember

$$\gamma_{i(k)}^* = \mathcal{P}(C = k \mid x_i; \Theta^*) \Rightarrow \hat{C}_i = \arg \max_k \text{softmax}(\gamma_{i(k)}^*)$$



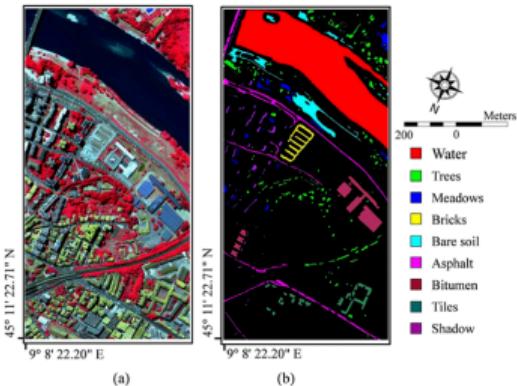


Gaussian Mixture Model

Unsupervised learning | Clustering

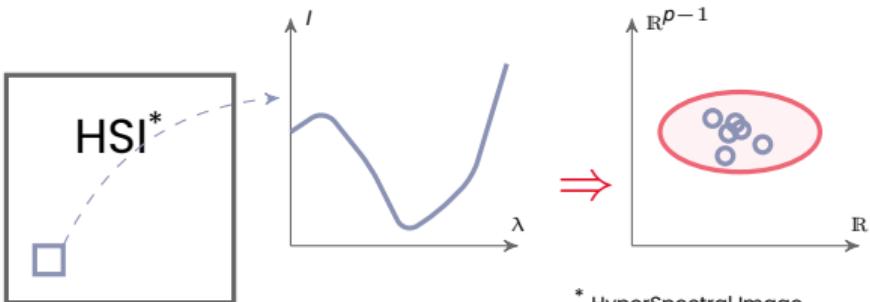
Remember

$$\gamma_{i(k)}^* = \mathcal{P}(C = k \mid x_i; \Theta^*) \Rightarrow \hat{C}_i = \arg \max_k \text{softmax}(\gamma_{i(k)}^*)$$



Zhao et al. (2016)

- can generalise to N -dimensional distributions
- used for **exploratory** data analysis...
- ...as well as unsupervised **classification**





3. ML Algorithms

Supervised learning

Table of contents

1. Some History and Philosophy to Start

2. The ML Mindset

3. ML Algorithms

Taxonomy of algorithms

Unsupervised learning

Supervised learning

Ensemble learning

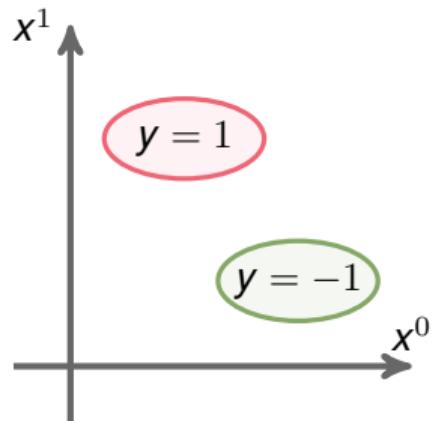
4. Neural Networks

5. Conclusions



Support Vector Machine

Definition | The classification case



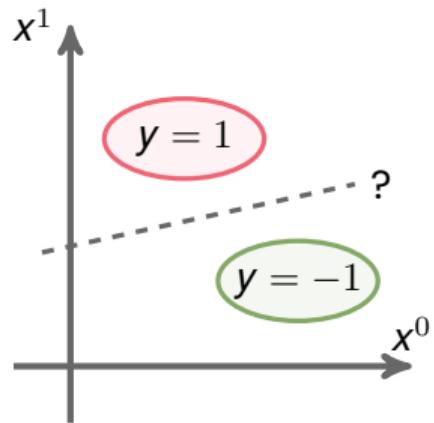
(see Cortes and Vapnik (1995))

Suppose $\mathcal{D} = \{(\vec{x}, y)\}$, s.t. $y = \pm 1$
(classification) (simple case: linearly separable)



Support Vector Machine

Definition | The classification case



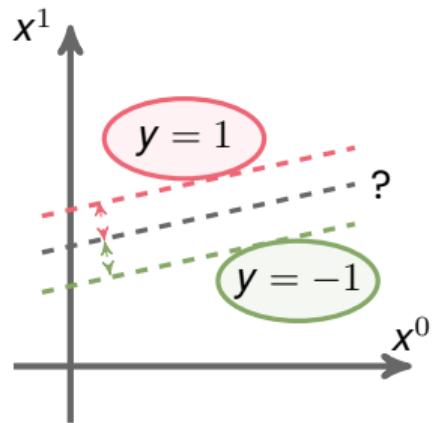
(see Cortes and Vapnik (1995))

How to choose the best
hyperplane to separate the
points?



Support Vector Machine

Definition | The classification case



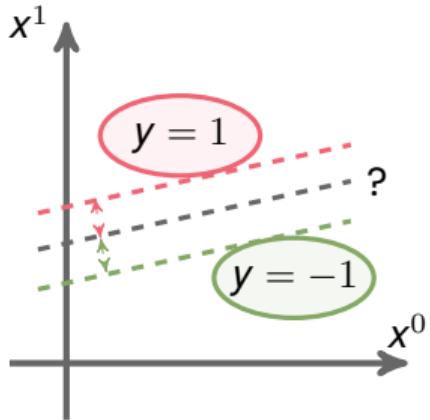
(see Cortes and Vapnik (1995))

Maximise the separation (i.e.
the **margin**)!



Support Vector Machine

Definition | The classification case



(see Cortes and Vapnik (1995))

Maximise the separation (i.e. the **margin**)!

Hyperplane (strict) separation theorem

Let $A, B \subset \mathbb{R}^p$ s.t. they are **closed** and **convex**, and $A \cap B = \emptyset$. Suppose one of them is **compact**. Then $\exists \vec{w} = (\vec{a}, b) \in \mathbb{R}^p \times \mathbb{R}$ s.t.

$$\vec{a} \cdot \vec{x} + b \begin{cases} > c_1 & \forall x \in A \\ < c_2 & \forall x \in B \end{cases}$$

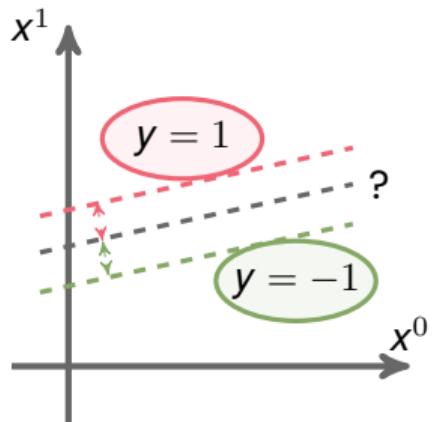
for $c_1 > c_2$. That is,

provided a separation, there exist a hyperplane separating the two sets



Support Vector Machine

Definition | The classification case



(see Cortes and Vapnik (1995))

Maximise the separation (i.e.
the **margin**)!

Hyperplane (strict) separation theorem

Let $A, B \subset \mathbb{R}^p$ s.t. they are **closed** and **convex**, and $A \cap B = \emptyset$. Suppose one of them is **compact**. Then $\exists \vec{w} = (\vec{a}, b) \in \mathbb{R}^p \times \mathbb{R}$ s.t.

$$\vec{a} \cdot \vec{x} + b \begin{cases} > c_1 & \forall x \in A \\ < c_2 & \forall x \in B \end{cases}$$

for $c_1 > c_2$. That is,

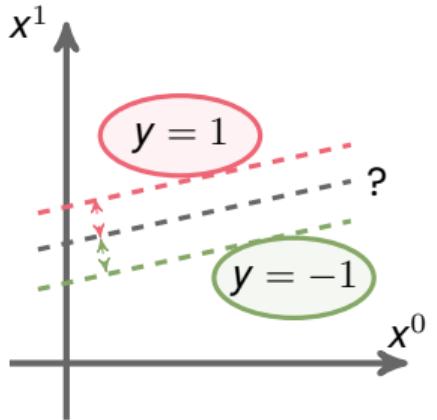
$$\begin{array}{rccc} \exists f : & \mathbb{R}^p & \rightarrow & \{-1, +1\} \\ & \vec{x} & \mapsto & y \end{array}$$

where y is a *boolean* variable.



Support Vector Machine

Definition | The classification case



(see Cortes and Vapnik (1995))

Support Vector Machine

Let $\mathcal{D} = \{(\vec{x}_i, y_i) \in \mathbb{R}^p \times \{-1, 1\} \mid i = 1, 2, \dots, n\}$

linearly separable. Let $(\vec{a}, b) \in \mathbb{R}^p \times \mathbb{R}$ identify a hyperplane.

Then, for $i = 1, 2, \dots, n$:

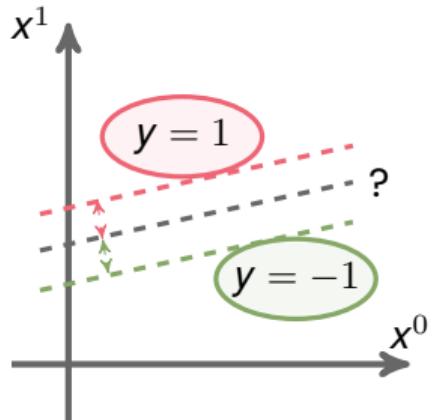
$$\vec{a} \cdot \vec{x}_i + b \begin{cases} \geq +1 & \text{if } y_i = +1 \\ \leq -1 & \text{if } y_i = -1 \end{cases}$$

Maximise the separation (i.e.
the **margin**)!



Support Vector Machine

Definition | The classification case



(see Cortes and Vapnik (1995))

Support Vector Machine

Let $\mathcal{D} = \{(\vec{x}_i, y_i) \in \mathbb{R}^p \times \{-1, 1\} \mid i = 1, 2, \dots, n\}$

linearly separable. Let $(\vec{a}, b) \in \mathbb{R}^p \times \mathbb{R}$ identify a hyperplane.

Then, for $i = 1, 2, \dots, n$:

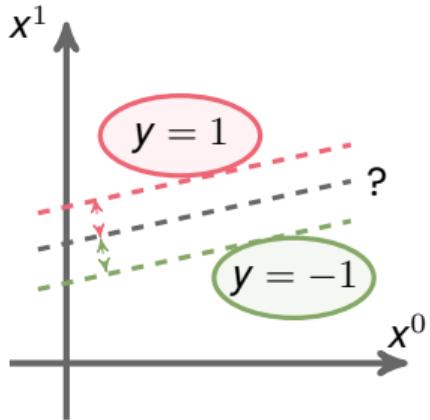
$$y_i (\vec{a} \cdot \vec{x}_i + b) \geq 1.$$

Maximise the separation (i.e.
the **margin**)!



Support Vector Machine

Definition | The classification case



(see Cortes and Vapnik (1995))

Maximise the separation (i.e.
the **margin**)!

Support Vector Machine

Let $\mathcal{D} = \{(\vec{x}_i, y_i) \in \mathbb{R}^p \times \{-1, 1\} \mid i = 1, 2, \dots, n\}$

linearly separable. Let $(\vec{a}, b) \in \mathbb{R}^p \times \mathbb{R}$ identify
a hyperplane.

The distance between classes

$$\rho(\vec{a}) = \min_{\vec{x}|y=+1} \frac{\vec{a}}{\|\vec{a}\|_2} \cdot \vec{x} - \max_{\vec{x}|y=-1} \frac{\vec{a}}{\|\vec{a}\|_2} \cdot \vec{x}$$

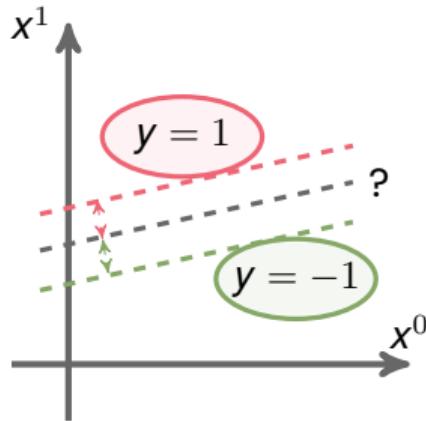
becomes maximal for the **optimal** (\vec{a}^*, b^*) :

$$\rho(\vec{a}^*) = \frac{2}{\|\vec{a}\|_2}.$$



Support Vector Machine

Definition | The classification case



Maximise the separation (i.e. the **margin**)!

Support Vector Machine

$$(\vec{a}^*, b^*) = \arg \max_{\vec{a} \in \mathbb{R}^p, b \in \mathbb{R}} \rho(\vec{a}) = \arg \min_{\vec{a} \in \mathbb{R}^p, b \in \mathbb{R}} \frac{1}{2} \|\vec{a}\|_2^2$$

constrained to

$$y_i(\vec{a} \cdot \vec{x}_i + b) \geq 1 \quad \forall i = 1, 2, \dots, n.$$

That is, find the **saddle points** of:

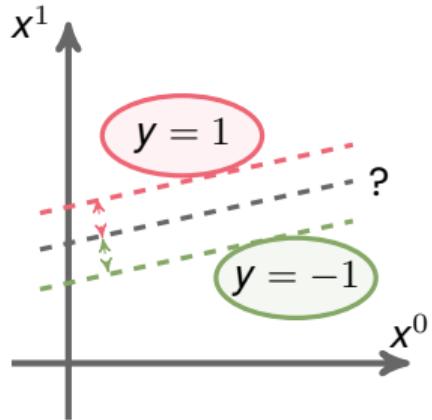
$$L(\vec{a}, b, \vec{\mu}) = \frac{1}{2} \vec{a} \cdot \vec{a} - \sum_{i=0}^{n-1} \mu_i (y_i(\vec{a} \cdot \vec{x}_i + b) - 1)$$

s.t. $\mu_i \geq 0 \quad \forall i = 1, 2, \dots, n$ (Lagrange multipliers \Rightarrow find the min for \vec{a} and b ,
and the max for $\vec{\mu}$).
AlPhy 30/09/2024 58 / 102



Support Vector Machine

Definition | The classification case



(see Cortes and Vapnik (1995))

Maximise the separation (i.e.
the **margin**)!

Support Vectors

Notice that $\mu_i \geq 0 \forall i = 1, 2, \dots, n$ is fundamental. From the minimisation (equation of motion), the constraint:

$$\mu_i (y_i(\vec{a} \cdot \vec{x}_i + b) - 1) = 0$$

implies that

$$\mu_i > 0 \Rightarrow y_i(\vec{a} \cdot \vec{x}_i + b) = 1$$

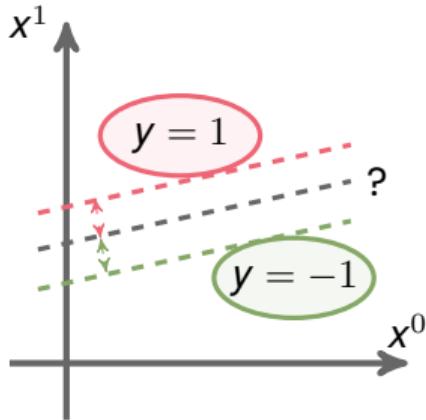
$\forall i = 1, 2, \dots, n$. That is,

the only contributing data points are those precisely on the margin \Rightarrow Support Vectors



Support Vector Machine

Definition | The classification case



(see Cortes and Vapnik (1995))

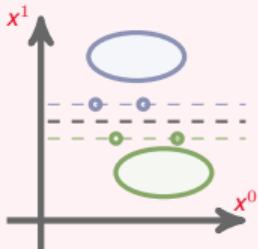
Maximise the separation (i.e.
the **margin**)!

Support Vector Machine

$$(\vec{a}^*, b^*) = \arg \min_{\vec{a} \in \mathbb{R}^P, b \in \mathbb{R}} \arg \max_{\vec{\mu} \in \mathbb{R}^n} L(\vec{a}, b, \vec{\mu})$$

can be performed to give:

$$\vec{a}^* = \sum_{i=1}^n y_i \mu_i \vec{x}_i \quad \text{and} \quad b^* = -\vec{a} \cdot \vec{x}_i.$$

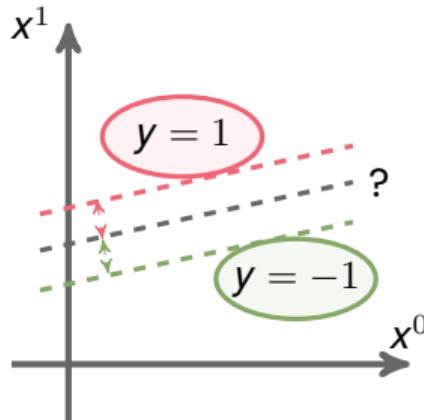


Hard Margin
Support Vector
Machine



Support Vector Machine

Definition | The classification case



(see Cortes and Vapnik (1995))

Maximise the separation (i.e.
the **margin**)!

Support Vector Machine (v2)

Suppose data is **not** perfectly separable. New
constraint:

$$y_i (\vec{a} \cdot \vec{x}_i + b) \geq 1 - \xi_i \quad \xi_i > 0 \quad \forall i = 1, 2, \dots, n.$$

Hence ($C > 0$):

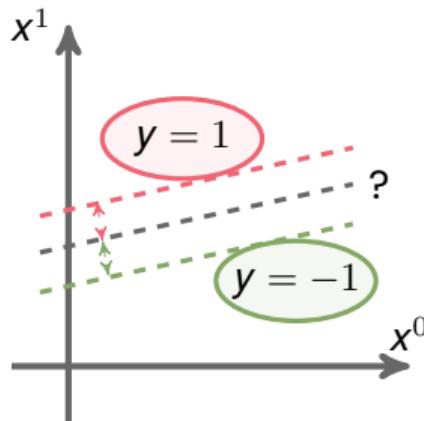
$$L(\vec{a}, b, \vec{\mu}, \vec{\xi}) = \underset{\vec{a} \in \mathbb{R}^p, b \in \mathbb{R}, \vec{\xi} \in \mathbb{R}^n}{\arg \min} \frac{1}{2} \vec{a} \cdot \vec{a} + C \sum_{i=1}^n \xi_i$$

Soft Margin SVM



Support Vector Machine

Definition | The classification case



(see Cortes and Vapnik (1995))

Maximise the separation (i.e.
the **margin**)!

Kernel SVM

Suppose data not (natively) linearly separable
in target space, but possibly in **feature space**:

$$\exists \phi: \mathbb{R}^p \rightarrow \mathbb{R}^P, \quad P > p$$

s.t. we can use SVM in P -dimensional space:

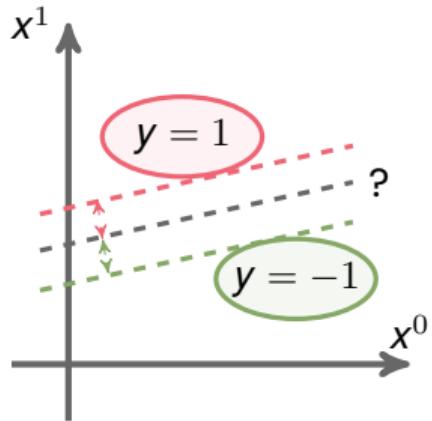
$$\vec{a}^* = \sum_{i=1}^n y_i \mu_i \phi(\vec{x}_i).$$

Do we need to know ϕ analytically?



Support Vector Machine

Definition | The classification case



(see Cortes and Vapnik (1995))

Maximise the separation (i.e.
the **margin**)!

Kernel SVM

Let f be the SVM classifier and \vec{x} a new data point:

$$\begin{aligned} f(\vec{x}) &= \phi(\vec{x}) \cdot \vec{a}^* + b^* \\ &= \sum_{i=1}^n y_i \mu_i \phi(\vec{x}) \cdot \phi(\vec{x}_i) + b^*. \end{aligned}$$

Only projections on *support vectors*:

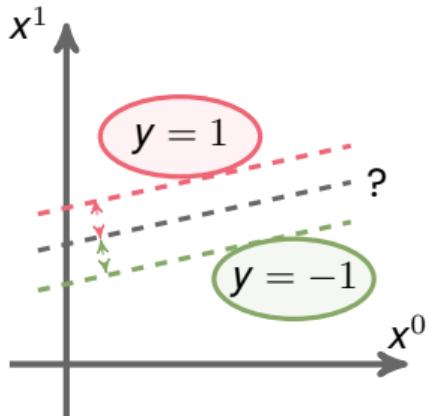
$$K(\vec{x}, \vec{x}_i) = \phi(\vec{x}) \cdot \phi(\vec{x}_i)$$

are really necessary (not even all data points!).



Support Vector Machine

Definition | The classification case



(see Cortes and Vapnik (1995))

Maximise the separation (i.e.
the **margin**)!

Kernel SVM

As long as $K \in L_2(\mathbb{R}^P)$ and symmetric:

$$K(\vec{u}, \vec{v}) = \sum_{i=1}^{\infty} \lambda_i \varphi_i(\vec{u}) \cdot \varphi_j(\vec{v}), \quad \lambda_i \geq 0.$$

Good/used choices of K :

- *radial basis func.:* $K_{\sigma}(\vec{u}, \vec{v}) = \exp(-\frac{\|\vec{u}-\vec{v}\|}{\sigma^2})$
- *polynomial func.:* $K_d(\vec{u}, \vec{v}) = (1 + \vec{u} \cdot \vec{v})^d$

HOMEWORK

1. build a SVM for regression (change the classifier constraint to a MSE loss, see Drucker et al. (1996))
2. show that the *sigmoid* can be used as a kernel function



Decision Trees

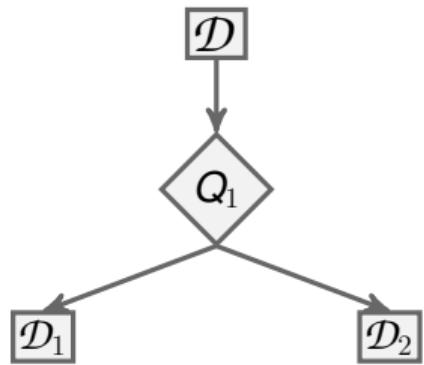
Definition

$\boxed{\mathcal{D}}$



Decision Trees

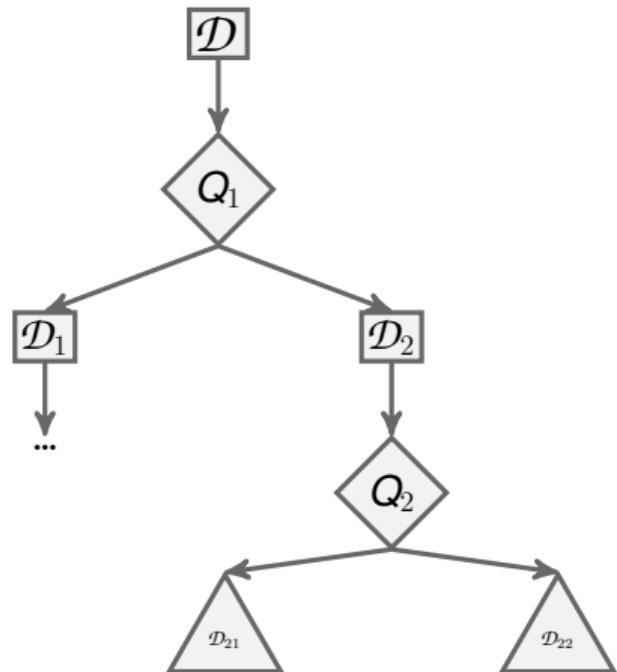
Definition





Decision Trees

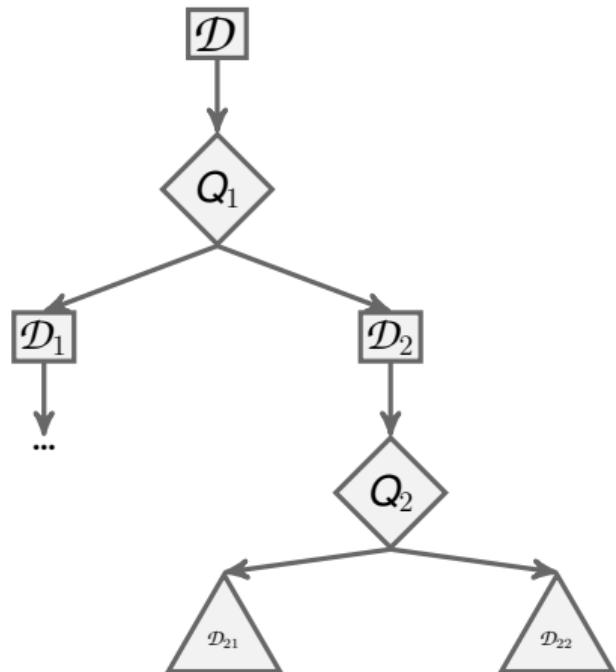
Definition





Decision Trees

Definition



A **decision tree** \mathcal{T} is a *hierarchical* model:

- **decision nodes** where “splits” are made
- **data nodes**
 - *branches* parent partitions of data
 - *leaves* final partitions of data

That is \mathcal{T} (slight abuse of notation: \mathcal{D} is both the data and the domain):

$$\mathcal{T} = \left\{ \mathcal{D}_A \subset \mathcal{D} \mid \bigcup_A \mathcal{D}_A = \mathcal{D}, \mathcal{D}_A \cap \mathcal{D}_B = \emptyset \text{ for } A \neq B \right\}$$

where

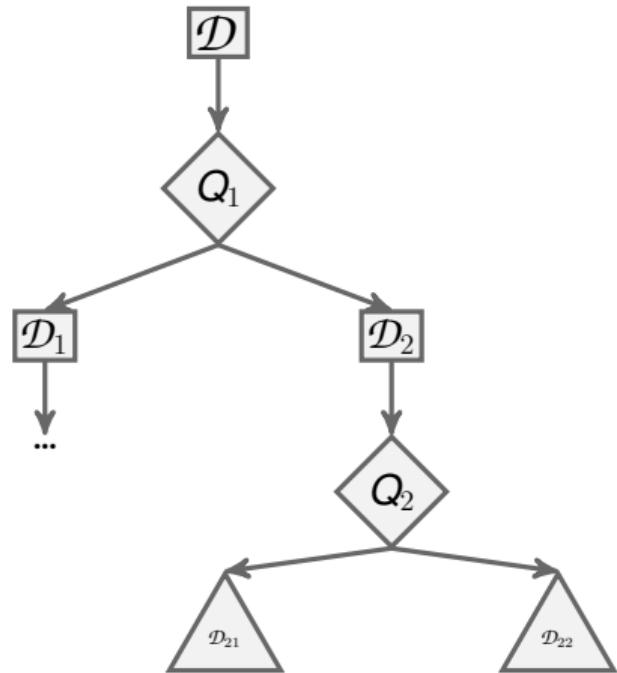
$$A = \{a_1, a_2, \dots, a_n\}$$

s.t. $|A|$ is maximal w.r.t. a *stopping criterion*.

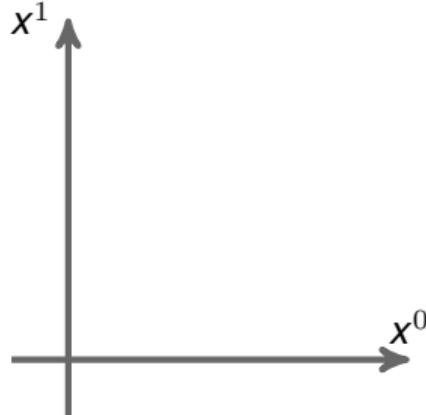


Decision Trees

Definition



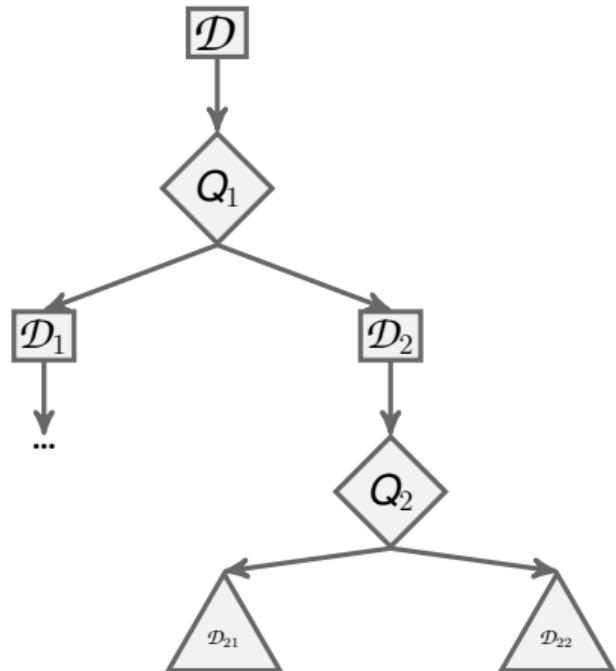
A **decision tree** \mathcal{T} is *piecewise linear* as it outputs a **tesselation** of the domain space:



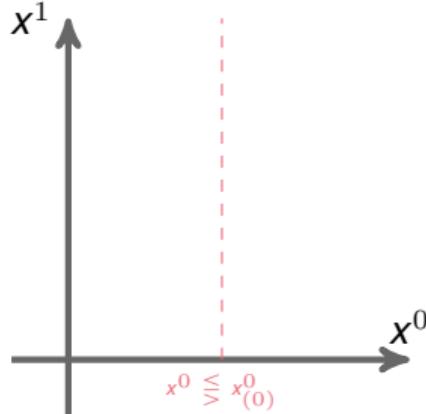


Decision Trees

Definition



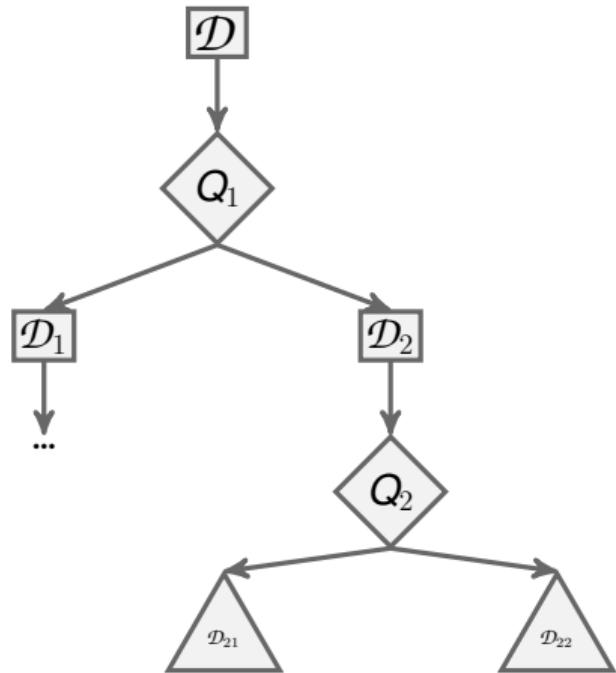
A **decision tree** \mathcal{T} is *piecewise linear* as it outputs a **tesselation** of the domain space:



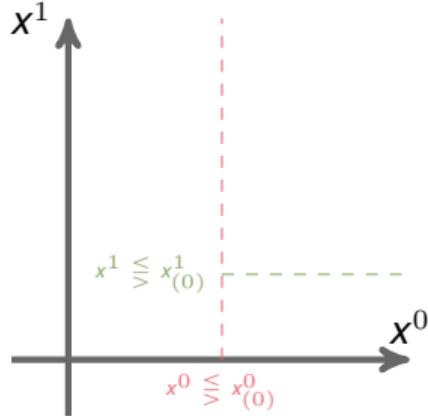


Decision Trees

Definition



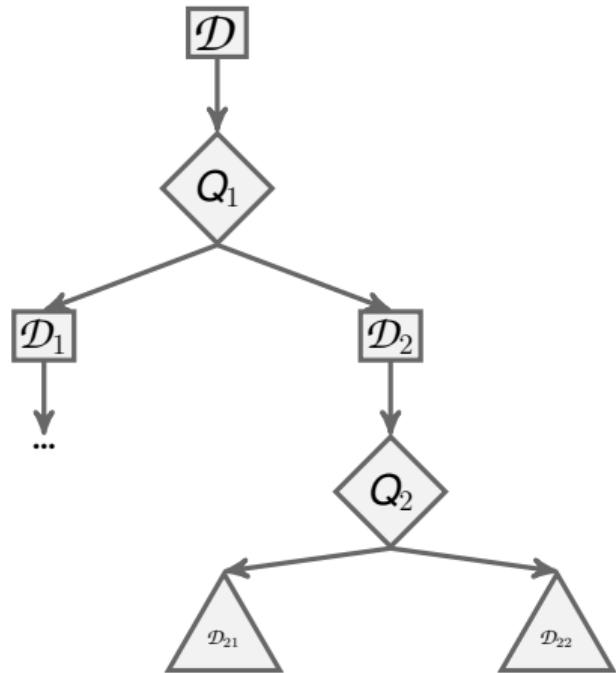
A **decision tree** \mathcal{T} is *piecewise linear* as it outputs a **tesselation** of the domain space:



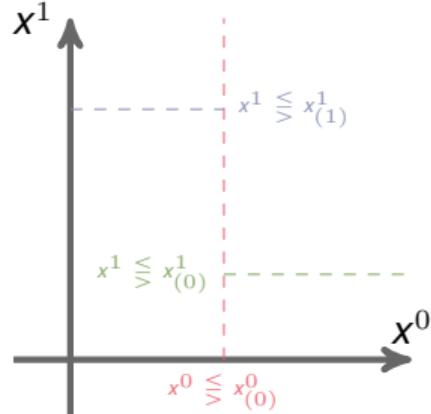


Decision Trees

Definition



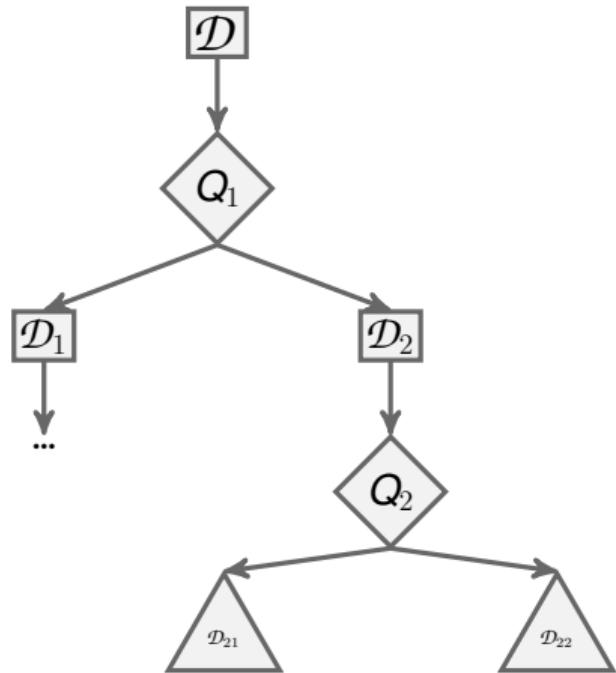
A **decision tree** \mathcal{T} is *piecewise linear* as it outputs a **tesselation** of the domain space:



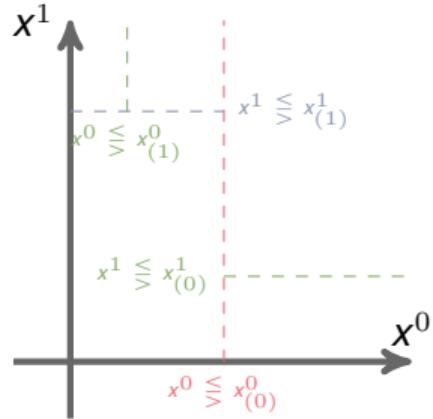


Decision Trees

Definition



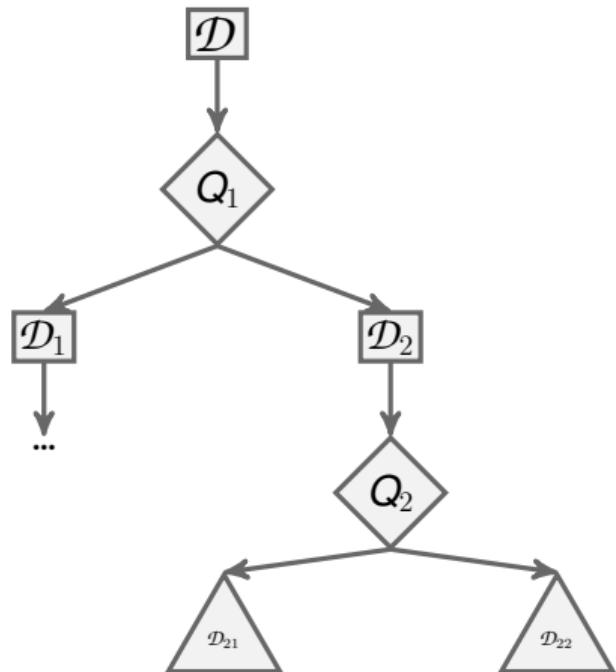
A **decision tree** \mathcal{T} is *piecewise linear* as it outputs a **tesselation** of the domain space:



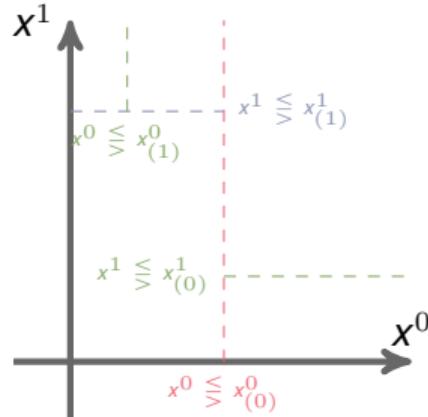


Decision Trees

Definition



A **decision tree** \mathcal{T} is *piecewise linear* as it outputs a **tesselation** of the domain space:

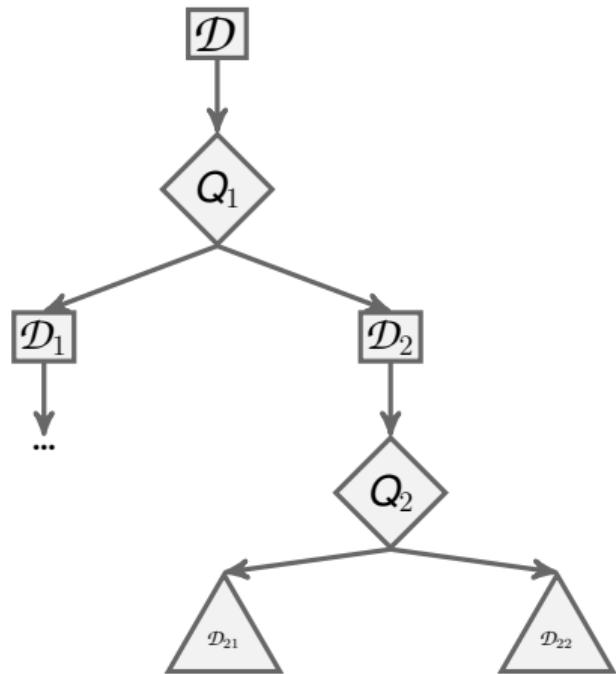


There exists a huge number of decision tree making algorithms (THAID, C4.5, CART, MARS, etc.) → we focus on CART and C4.5.



Decision Trees

Definition



In other words:

Require: stopping criterion \mathcal{S} , measure of "goodness of split" \mathcal{G} , $i \leftarrow 0$

while $\neg \mathcal{S}$ **do**

loop

select node i

find the best partition of \mathcal{D}_i according to \mathcal{G}

create child nodes i_a ($a = i_1, i_2, \dots$)

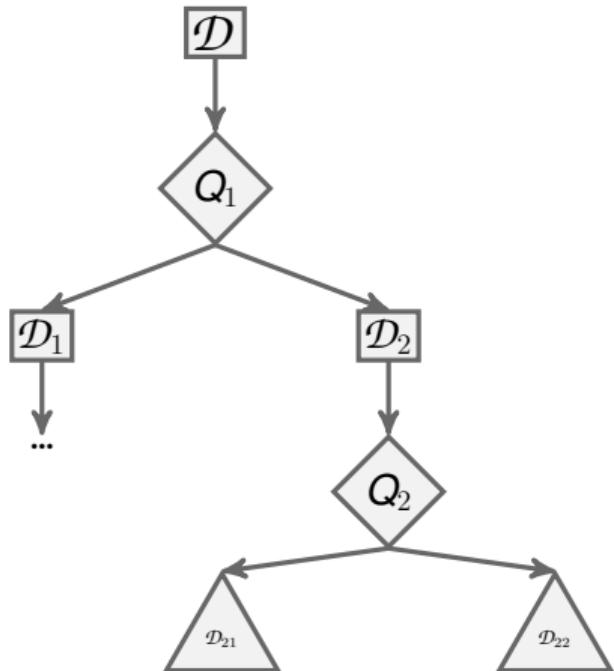
loop for each child node $i \leftarrow i_a$

return partition assignments of \mathcal{D}



Decision Trees

Definition



The **C4.5** decision tree

Use **information gain** (mutual information) as criterion \mathcal{G} (maximise):

$$\text{MI}(X \in \mathcal{D}, X_i \in \mathcal{D}_A) = H(X \in \mathcal{D}) - H(X \in \mathcal{D} \mid X_i \in \mathcal{D}_A),$$

where

$$H(X) = - \sum_{j=1}^K P(X \in C_j) \log_2 P(X \in C_j),$$

$$H(X \mid X_i) = - \sum_{j=1}^K P(X \wedge X_i) \log_2 \frac{P(X \wedge X_i)}{P(X \in C_j)},$$

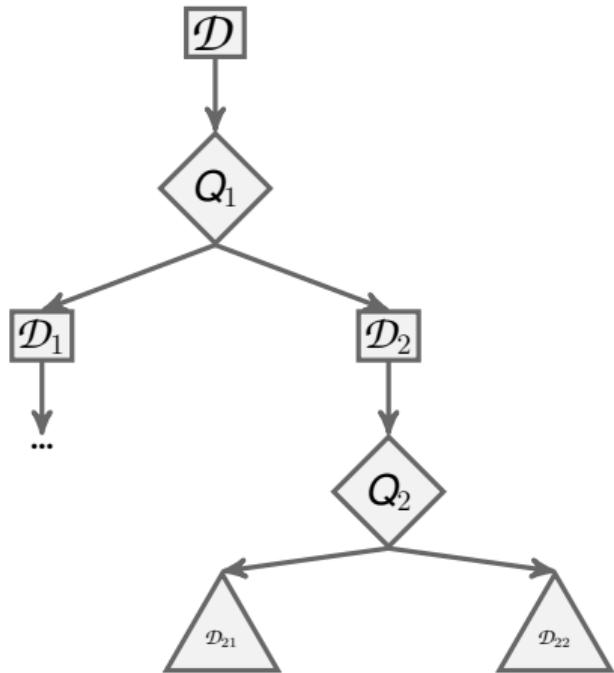
and $P(X \wedge X_i) = P(X \in \mathcal{D}, X_i \in C_j)$ and $C_j \subset \mathcal{D}$.

(see Quinlan (1994))



Decision Trees

Definition



The C4.5 decision tree

Use **information gain** (mutual information) as criterion \mathcal{G} (maximise):

$$MI(X \in \mathcal{D}, X_i \in \mathcal{D}_A) = H(X \in \mathcal{D}) - H(X \in \mathcal{D} | X_i \in \mathcal{D}_A),$$

Which implies:

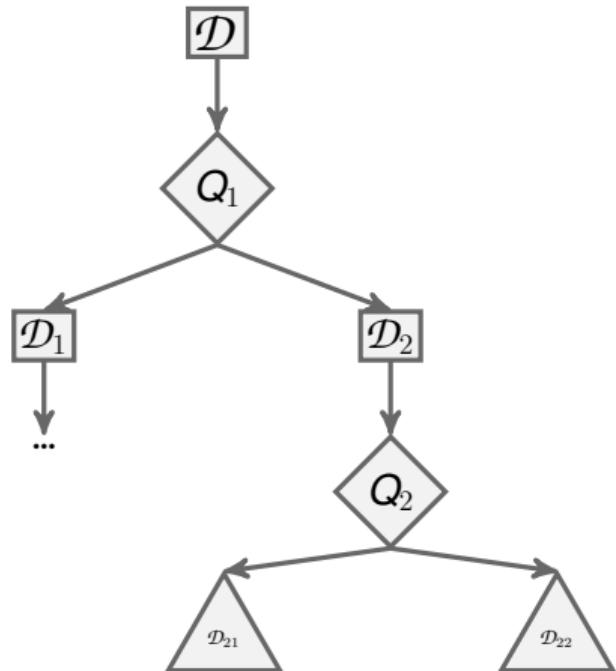
- **maximise** the information acquired by the split
- **pruning** based on informative splits
(uninformative branches are replaced by leaf nodes)
- **missing values** automatically handled
- the split can be **arbitrary** (e.g. multiclass)

(see Quinlan (1994))



Decision Trees

Definition



The Classification And Regression Trees

Use **Gini impurity** as \mathcal{G} (minimise).

Let $p_i, i = 1, 2, \dots, K$, be the probability of choosing an item of class C_i :

$$I(X \in \mathcal{D}_A) = \sum_{i=1}^K p_i(1 - p_i) = 1 - \sum_{i=1}^K p_i^2,$$

that is

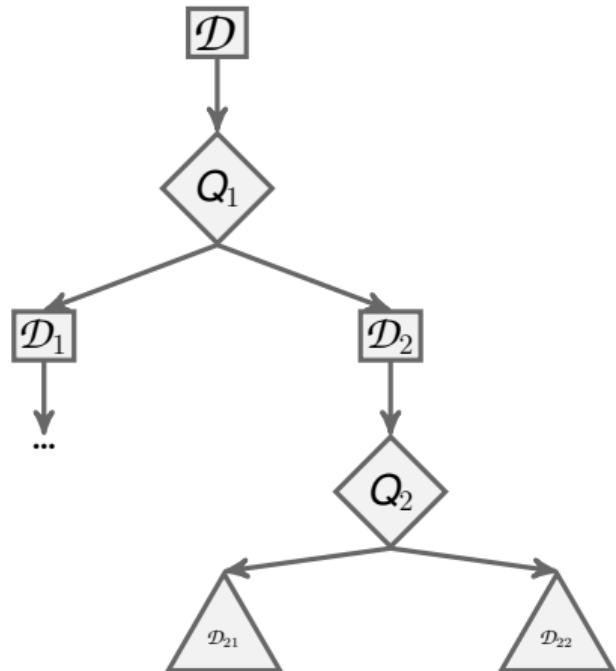
the probability of incorrectly classifying an item, if it were randomly labelled based on the distribution of the sample.

(see Breiman et al. (1984))



Decision Trees

Definition



The Classification And Regression Trees

Use **Gini impurity** as \mathcal{G} (minimise).
Let $p_i, i = 1, 2, \dots, K$, be the probability of choosing an item of class C_i :

$$I(X \in \mathcal{D}_A) = \sum_{i=1}^K p_i(1 - p_i) = 1 - \sum_{i=1}^K p_i^2,$$

that is

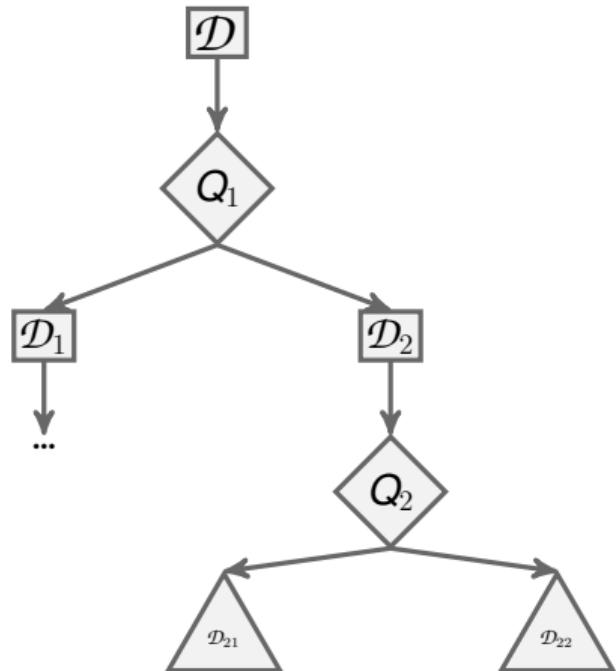
the **Tsallis entropy** (generalised Boltzmann-Gibbs) with deformation 2.

(see Breiman et al. (1984))



Decision Trees

Definition



The Classification And Regression Trees

Use **Gini impurity** as \mathcal{G} (minimise).
Let $p_i, i = 1, 2, \dots, K$, be the probability of choosing an item of class C_i :

$$I(X \in \mathcal{D}_A) = \sum_{i=1}^K p_i(1 - p_i) = 1 - \sum_{i=1}^K p_i^2,$$

which enables:

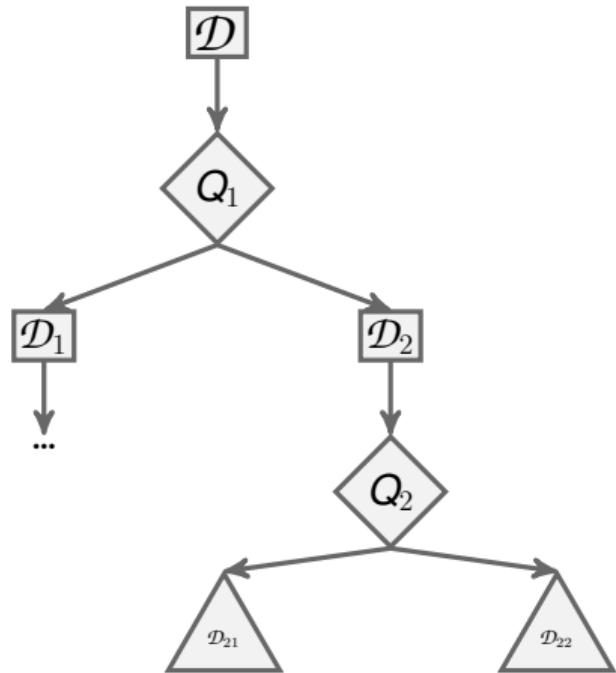
- **binary** partitions of \mathcal{D}_A
- **pruning** to be enforced (e.g. cross-validation)
- **label = mode** of leaf node (*mean/median*)

(see Breiman et al. (1984))



Decision Trees

Definition



THINK

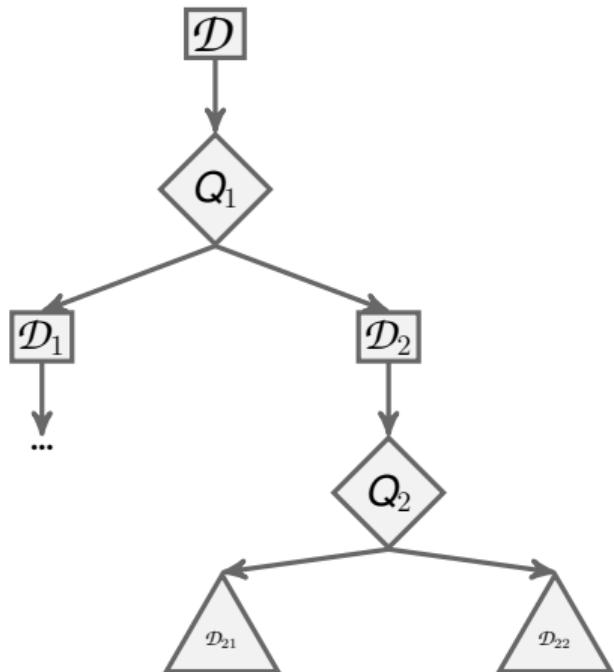
Suppose no pruning:

1. at which point does \mathcal{T} naturally stop?
2. is \mathcal{T} **high bias** or **high variance**?



Decision Trees

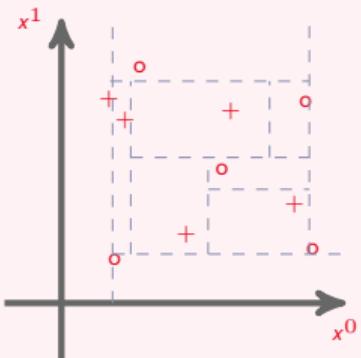
Definition



THINK

Suppose no pruning:

1. at which point does \mathcal{T} naturally stop? $\rightarrow |\mathcal{D}_A| = 1, \forall A$
2. is \mathcal{T} high bias or high variance? $\rightarrow \text{VERY high variance}$

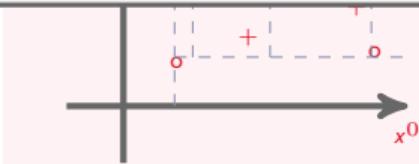
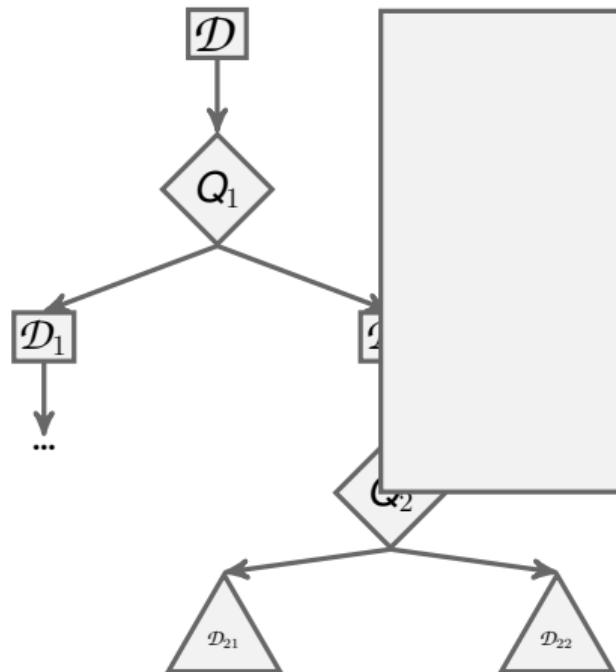


Decision trees are prone to **overfitting**
 $\mathcal{D}^{(\text{train})}$ without appropriate strategies!



Decision Trees

Definition



$|\mathcal{D}_A| = 1, \forall A$

RY high variance

Decision trees are one to **overfitting** $\mathcal{D}^{(train)}$ without appropriate strategies!



Decision Trees

Feature ranking

Hierarchical structure enables **ranking** features \Rightarrow **importance** of feature for the split.



Decision Trees

Feature ranking

Hierarchical structure enables **ranking** features \Rightarrow **importance** of feature for the split.

Let

$$\mathfrak{I}_A = \frac{|\mathcal{D}_A|}{|\mathcal{D}|} \mathbf{1}(X \in \mathcal{D}_A) - \frac{|\mathcal{D}_{A1}|}{|\mathcal{D}|} \mathbf{1}(X \in \mathcal{D}_{A1}) - \frac{|\mathcal{D}_{A2}|}{|\mathcal{D}|} \mathbf{1}(X \in \mathcal{D}_{A2})$$

the importance of node A .



Decision Trees

Feature ranking

Hierarchical structure enables **ranking** features \Rightarrow **importance** of feature for the split.

Let

$$\mathfrak{I}_A = \frac{|\mathcal{D}_A|}{|\mathcal{D}|} \mathbf{1}(X \in \mathcal{D}_A) - \frac{|\mathcal{D}_{A1}|}{|\mathcal{D}|} \mathbf{1}(X \in \mathcal{D}_{A1}) - \frac{|\mathcal{D}_{A2}|}{|\mathcal{D}|} \mathbf{1}(X \in \mathcal{D}_{A2})$$

the importance of node A .

Compute the **feature importance** of feature i :

$$F_i = \frac{\sum_a \text{splits on } i \mathfrak{I}_a}{\sum_a \mathfrak{I}_a}$$



Decision Trees

Feature ranking

Hierarchical structure enables **ranking** features \Rightarrow **importance** of feature for the split.

```
● ● ●  
1 from sklearn.datasets import load_iris  
2 from sklearn.model_selection import train_test_split  
3 from sklearn.tree import DecisionTreeClassifier  
4 from sklearn.metrics import precision_recall_fscore_support  
5  
6 # Load and split the data  
7 iris = load_iris()  
8 X_train, X_test, y_train, y_test = train_test_split(iris.data,  
9 iris.target,  
10 shuffle=True,  
11 random_state=42)  
12  
13 # Train a decision tree  
14 clf = DecisionTreeClassifier().fit(X_train, y_train)  
15  
16 # Predict the test set  
17 y_test_pred = clf.predict(X_test)  
18 prec, rec, f1, _ = precision_recall_fscore_support(y_test,  
19 y_test_pred,  
20 average='macro')  
21 print(f'Precision: {prec:.0%}')  
22 print(f'Recall: {rec:.0%}')  
23 print(f'F1: {f1:.0%}')
```

```
● ● ●  
1 from sklearn.tree import plot_tree  
2  
3 plot_tree(clf,  
4             feature_names=iris.feature_names,  
5             class_names=iris.target_names,  
6             filled=True)
```

Can you produce/guess the output?

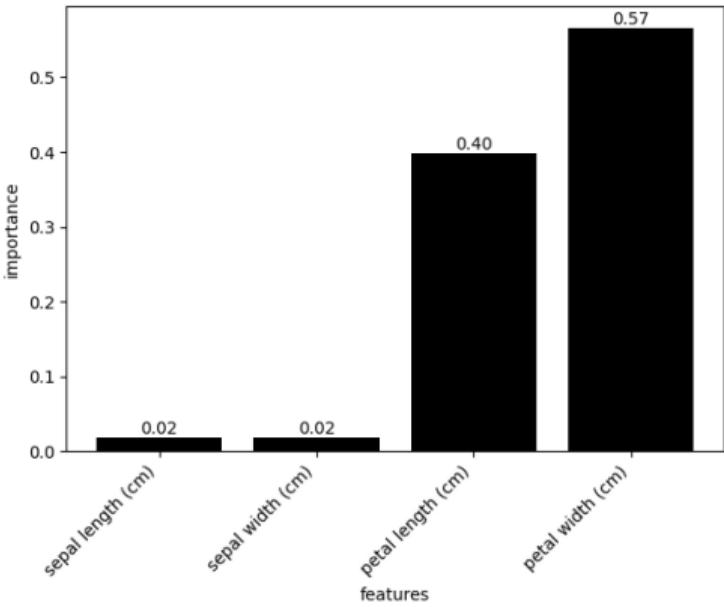




Decision Trees

Feature ranking

Hierarchical structure enables **ranking** features \Rightarrow **importance** of feature for the split.





3. ML Algorithms

Ensemble learning

Table of contents

1. Some History and Philosophy to Start

2. The ML Mindset

3. ML Algorithms

Taxonomy of algorithms

Unsupervised learning

Supervised learning

Ensemble learning

4. Neural Networks

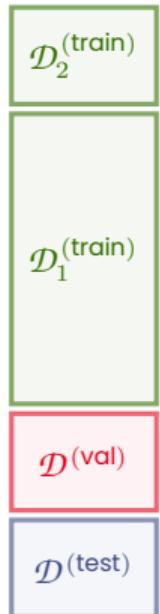
5. Conclusions



Ensemble Learning

Stacking / Metalearning

Let $\mathcal{D}_1^{(\text{train})} \cup \mathcal{D}_2^{(\text{train})} = \mathcal{D}^{(\text{train})}$ a partition of the training set:

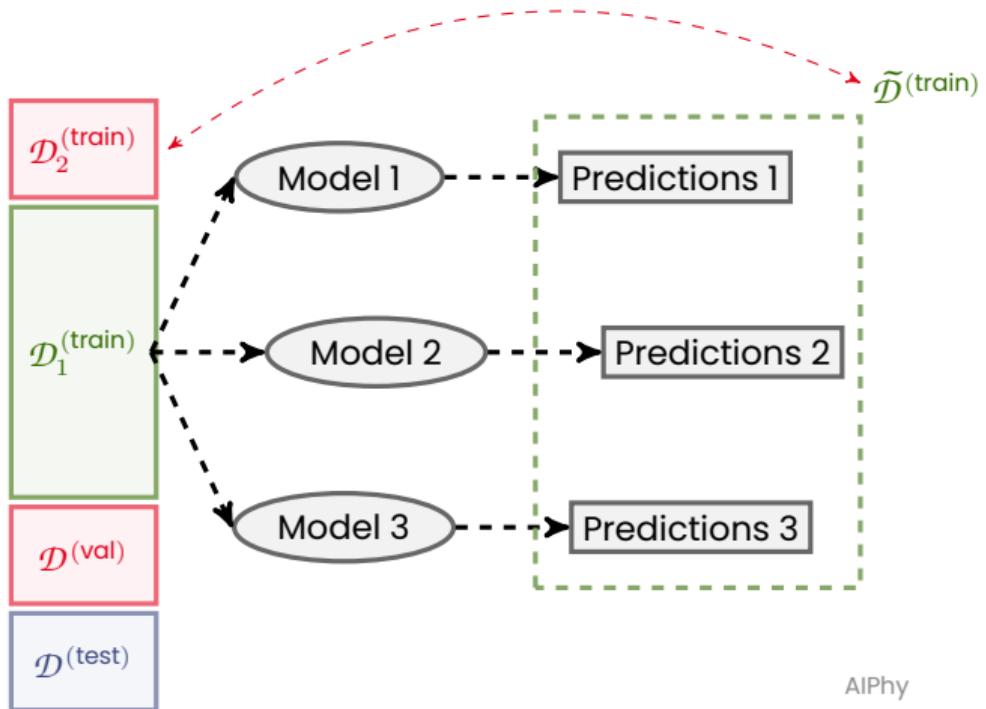




Ensemble Learning

Stacking / Metalearning

Let $\mathcal{D}_1^{(\text{train})} \cup \mathcal{D}_2^{(\text{train})} = \mathcal{D}^{(\text{train})}$ a partition of the training set:

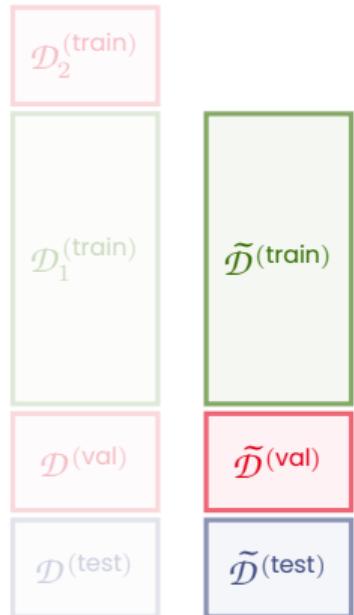




Ensemble Learning

Stacking / Metalearning

Let $\mathcal{D}_1^{(\text{train})} \cup \mathcal{D}_2^{(\text{train})} = \mathcal{D}^{(\text{train})}$ a partition of the training set:

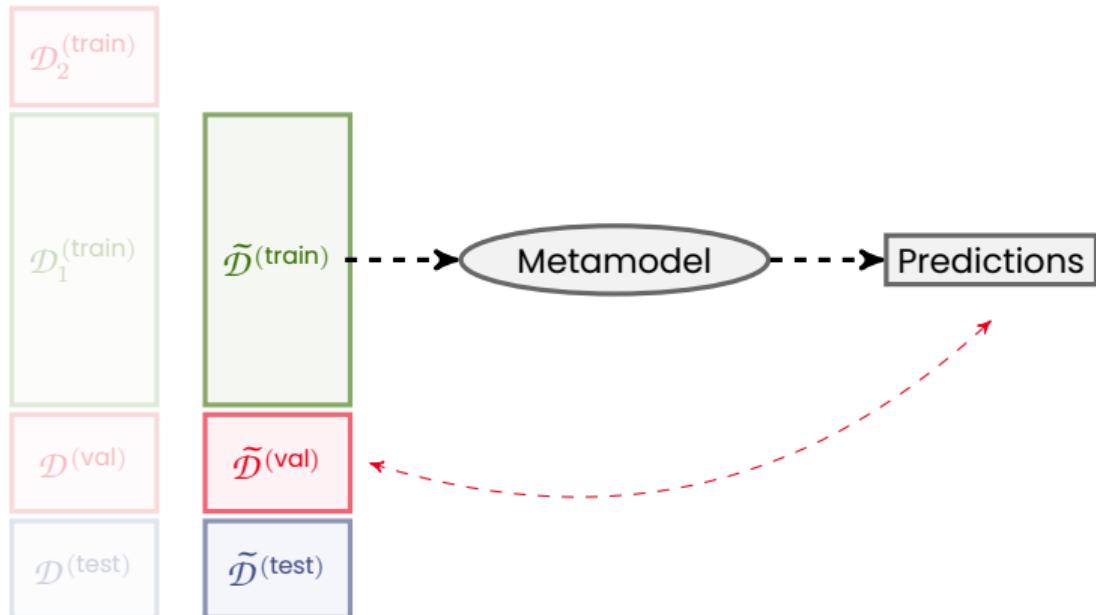




Ensemble Learning

Stacking / Metalearning

Let $\mathcal{D}_1^{(\text{train})} \cup \mathcal{D}_2^{(\text{train})} = \mathcal{D}^{(\text{train})}$ a partition of the training set:

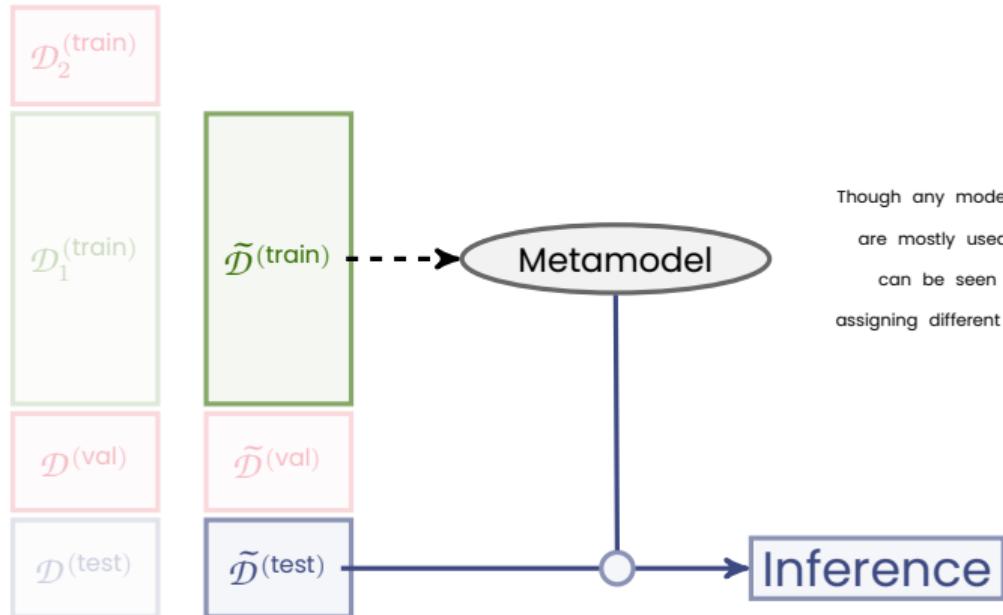




Ensemble Learning

Stacking / Metalearning

Let $\mathcal{D}_1^{(\text{train})} \cup \mathcal{D}_2^{(\text{train})} = \mathcal{D}^{(\text{train})}$ a partition of the training set:



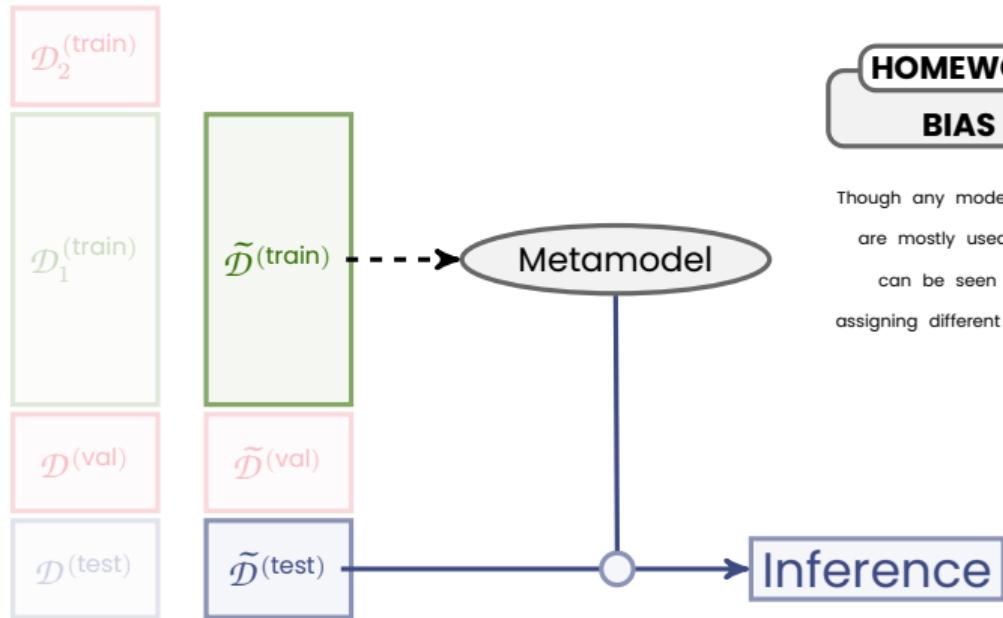
Though any model will do, *linear regression/logistic classification* are mostly used. Thanks to their simple interpretation, they can be seen as a generalisation of *majority voting*, by assigning different weights to the predictions of different models



Ensemble Learning

Stacking / Metalearning

Let $\mathcal{D}_1^{(\text{train})} \cup \mathcal{D}_2^{(\text{train})} = \mathcal{D}^{(\text{train})}$ a partition of the training set:



HOMEWORK

BIAS VS VARIANCE?

Though any model will do, *linear regression/logistic classification* are mostly used. Thanks to their simple interpretation, they can be seen as a generalisation of *majority voting*, by assigning different weights to the predictions of different models



Ensemble Learning

Bootstrap

Suppose a population with distribution \mathcal{P} for which you need a **statistical estimate** with expected value θ , and variance σ^2 :

1. take a sample $\mathcal{D} = \{X_1 = x_1, \dots, X_n = x_n\}$ with distribution $\hat{\mathcal{P}}$
2. estimate $\hat{\theta}$ using \mathcal{D}



Ensemble Learning

Bootstrap

Suppose a population with distribution \mathcal{P} for which you need a **statistical estimate** with expected value θ , and variance σ^2 :

1. take a sample $\mathcal{D} = \{X_1 = x_1, \dots, X_n = x_n\}$ with distribution $\widehat{\mathcal{P}}$
2. estimate $\widehat{\theta}$ using \mathcal{D}

Should you be able to repeat your estimation, you could compute

$$\mathbb{E}_{\widehat{\mathcal{P}}(X)}[\widehat{\theta}] = \theta \text{ (unbiased estimator)} \quad \text{Var}_{\widehat{\mathcal{P}}(X)}[\widehat{\theta}] = \frac{n-1}{n} \sigma^2 \text{ (biased estimator)}$$



Ensemble Learning

Bootstrap

Suppose a population with distribution \mathcal{P} for which you need a **statistical estimate** with expected value θ , and variance σ^2 :

1. take a sample $\mathcal{D} = \{X_1 = x_1, \dots, X_n = x_n\}$ with distribution $\widehat{\mathcal{P}}$
2. estimate $\widehat{\theta}$ using \mathcal{D}

Should you be able to repeat your estimation, you could compute

$$\mathbb{E}_{\widehat{\mathcal{P}}(X)}[\widehat{\theta}] = \theta \text{ (unbiased estimator)} \quad \text{Var}_{\widehat{\mathcal{P}}(X)}[\widehat{\theta}] = \frac{n-1}{n}\sigma^2 \text{ (biased estimator)}$$

and show

$$\frac{\mathbb{E}_{\widehat{\mathcal{P}}(X)}[\widehat{\theta}] - \theta}{\frac{\sigma}{\sqrt{n}}} \sim \mathcal{N}(0, 1) \text{ (central limit theorem).}$$



Ensemble Learning

Bootstrap

Suppose a population with distribution \mathcal{P} for which you need a **statistical estimate** with expected value θ , and variance σ^2 :

1. take a sample $\mathcal{D} = \{X_1 = x_1, \dots, X_n = x_n\}$ with distribution $\hat{\mathcal{P}}$
2. estimate $\hat{\theta}$ using \mathcal{D}

Should you be able to repeat your estimation, you could compute

$$\mathbb{E}_{\hat{\mathcal{P}}(X)}[\hat{\theta}] = \theta \text{ (unbiased estimator)} \quad \text{Var}_{\hat{\mathcal{P}}(X)}[\hat{\theta}] = \frac{n-1}{n}\sigma^2 \text{ (biased estimator)}$$

and show

$$\frac{\mathbb{E}_{\hat{\mathcal{P}}(X)}[\hat{\theta}] - \theta}{\frac{\sigma}{\sqrt{n}}} \sim \mathcal{N}(0, 1) \text{ (central limit theorem).}$$

This might not be possible!



Ensemble Learning

Bootstrap

From the sample \mathcal{D} , you can *resample with replacement*:

"bootstrap" $\rightarrow \begin{cases} \mathcal{D}_{(1)}^* = \{x_1^*, x_2^*, \dots, x_n^*\} \longrightarrow \theta_{(1)}^* \\ \mathcal{D}_{(2)}^* = \{x_1^*, x_2^*, \dots, x_n^*\} \longrightarrow \theta_{(2)}^* \\ \dots \\ \mathcal{D}_{(B)}^* = \{x_1^*, x_2^*, \dots, x_n^*\} \longrightarrow \theta_{(B)}^* \end{cases}$



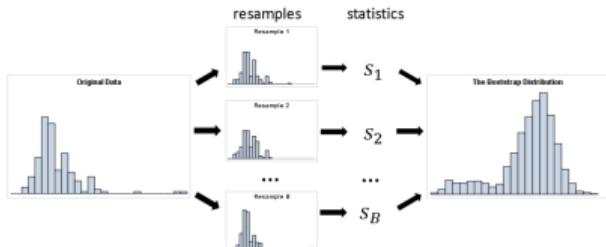


Ensemble Learning

Bootstrap

From the sample \mathcal{D} , you can *resample with replacement*:

"bootstrap" $\rightarrow \begin{cases} \mathcal{D}_{(1)}^* = \{x_1^*, x_2^*, \dots, x_n^*\} \longrightarrow \theta_{(1)}^* \\ \mathcal{D}_{(2)}^* = \{x_1^*, x_2^*, \dots, x_n^*\} \longrightarrow \theta_{(2)}^* \\ \dots \\ \mathcal{D}_{(B)}^* = \{x_1^*, x_2^*, \dots, x_n^*\} \longrightarrow \theta_{(B)}^* \end{cases}$



Bootstrap

Let $\widehat{\theta}^* = \mathbb{E}_{\mathcal{P}^*(X)}[\theta^*]$, where $\mathcal{P}^*(X)$ is the *bootstrap distribution*:

$$\widehat{\theta}^* \xrightarrow{P} \widehat{\theta}.$$



Ensemble Learning

Bootstrap

(see also [Chen \(2019\), Washington U.](#))

Consider the *Monte Carlo* bootstrap estimate

$$\text{Var}_{\mathcal{P}^*(X)}(\hat{\theta}^*) = \frac{1}{B-1} \sum_{i=1}^B \left(\hat{\theta}_{(i)}^* - \mathbb{E}[\hat{\theta}^*] \right) \xrightarrow{B \gg 1} \text{Var}_{\mathcal{P}^*(X|\mathcal{D})}(\hat{\theta}^*) \text{ ("with the sample } \mathcal{D} \text{ fixed").}$$

and prove

$$\text{Var}_{\mathcal{P}^*(X|\mathcal{D})}(\hat{\theta}^*) \xrightarrow{\text{P}} \text{Var}_{\widehat{\mathcal{P}}(X)}(\hat{\theta})$$



Ensemble Learning

Bootstrap

(see also Chen (2019), Washington U.)

Consider the *Monte Carlo* bootstrap estimate

$$\text{Var}_{\mathcal{P}^*(X)}(\hat{\theta}^*) = \frac{1}{B-1} \sum_{i=1}^B \left(\hat{\theta}_{(i)}^* - \mathbb{E}[\hat{\theta}^*] \right) \xrightarrow{B \gg 1} \text{Var}_{\mathcal{P}^*(X|\mathcal{D})}(\hat{\theta}^*) \text{ ("with the sample } \mathcal{D} \text{ fixed").}$$

and prove

$$\text{Var}_{\mathcal{P}^*(X|\mathcal{D})}(\hat{\theta}^*) \xrightarrow{\text{P}} \text{Var}_{\widehat{\mathcal{P}}(X)}(\hat{\theta})$$

Sketch of the proof:

Let $\Delta^*(\mathcal{D}, B) = \text{Var}_{\mathcal{P}^*(X|\mathcal{D})}(\hat{\theta}^*) - \text{Var}_{\widehat{\mathcal{P}}}(\hat{\theta})$, and suppose $\Delta^*(\mathcal{D}, B) < \frac{c}{B}$, with $c > 0$:

- (McDiarmid's inequality) $\text{P}(|\Delta^*(\mathcal{D}, B) - \mathbb{E}[\Delta^*(\mathcal{D}, B)]| \geq \varepsilon) \leq 2e^{-\frac{2\varepsilon^2\sqrt{B}}{c}}$
- (Borel-Cantelli lemma) $\sum_{i=1}^B \text{P}(|\Delta^*(\mathcal{D}, B) - \mathbb{E}[\Delta^*(\mathcal{D}, B)]| \geq \varepsilon) < \infty \Rightarrow \text{P}\left(\limsup_{B \rightarrow \infty} |\Delta^*(\mathcal{D}, B) - \mathbb{E}[\Delta^*(\mathcal{D}, B)]| \geq \varepsilon\right) = 0$
- $\mathbb{E}[\Delta^*(\mathcal{D}, B)] = 0 \Rightarrow \text{Var}_{\mathcal{P}^*(X|\mathcal{D})}\hat{\theta}^* \xrightarrow{\text{P}} \text{Var}_{\widehat{\mathcal{P}}(X)}\hat{\theta}$

□



Ensemble Learning

Bootstrap | The bootstrap theorem

With all these elements:

$$\left. \begin{array}{l} Z = \sqrt{n} \frac{\hat{\theta} - \theta}{\sqrt{\text{Var}_{P(X)}(\theta)}} \sim \mathcal{N}(0, 1) \\ Z^* = \sqrt{n} \frac{\hat{\theta}^* - \hat{\theta}}{\sqrt{\text{Var}_{\hat{P}(X)}(\theta)}} \sim \mathcal{N}(0, 1) \end{array} \right\} \text{estimated parameters have the same distribution!}$$



Ensemble Learning

Bootstrap | The bootstrap theorem

With all these elements:

$$\left. \begin{array}{l} Z = \sqrt{n} \frac{\hat{\theta} - \theta}{\sqrt{\text{Var}_{\mathcal{P}(X)}(\theta)}} \sim \mathcal{N}(0, 1) \\ Z^* = \sqrt{n} \frac{\hat{\theta}^* - \hat{\theta}}{\sqrt{\text{Var}_{\widehat{\mathcal{P}}(X)}(\theta)}} \sim \mathcal{N}(0, 1) \end{array} \right\} \text{estimated parameters have the same distribution!}$$

Slightly more formally:

Let

$$P(x) = P(Z \leq x) \quad P^*(x) = P(Z^* \leq x),$$

then (see Berry-Esseen theorem):

$$|P(x) - P^*(x)| \leq |P(x) - \Phi(x)| + |\Phi(x) - \Phi^*(x)| + |P^*(x) - \Phi^*(x)| \leq C \frac{\mu_3}{\sigma_3 \sqrt{n}} + O\left(n^{-\frac{1}{2}}\right) + C^* \frac{\mu_3^*}{\sigma_3^* \sqrt{n}} \rightarrow 0$$

when $n \rightarrow \infty$. This shows $P \rightarrow P^*$ in distribution.

□



Ensemble Learning

Bootstrap + Aggregating = “Bagging”

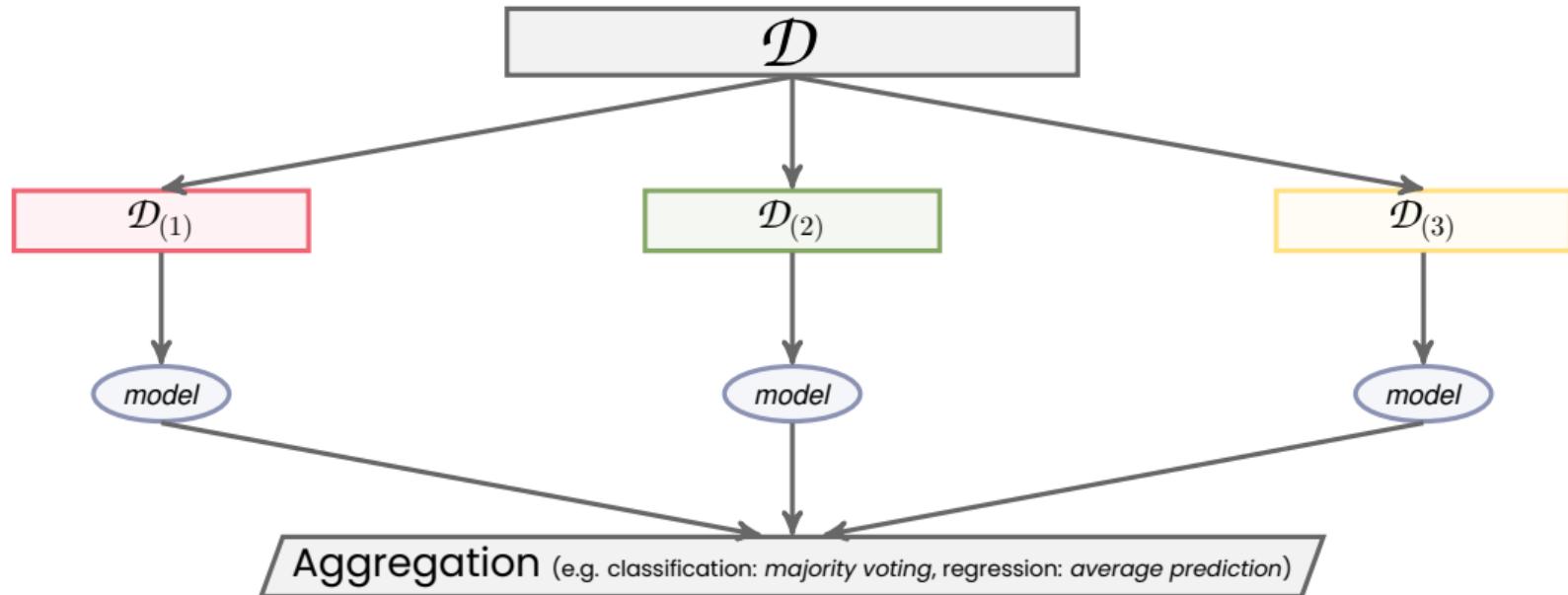
Given the previous discussion, it becomes natural to define the **bagging**:



Ensemble Learning

Bootstrap + Aggregating = "Bagging"

Given the previous discussion, it becomes natural to define the **bagging**:



more on this later...



Ensemble Learning

Boosting

Strong Learner

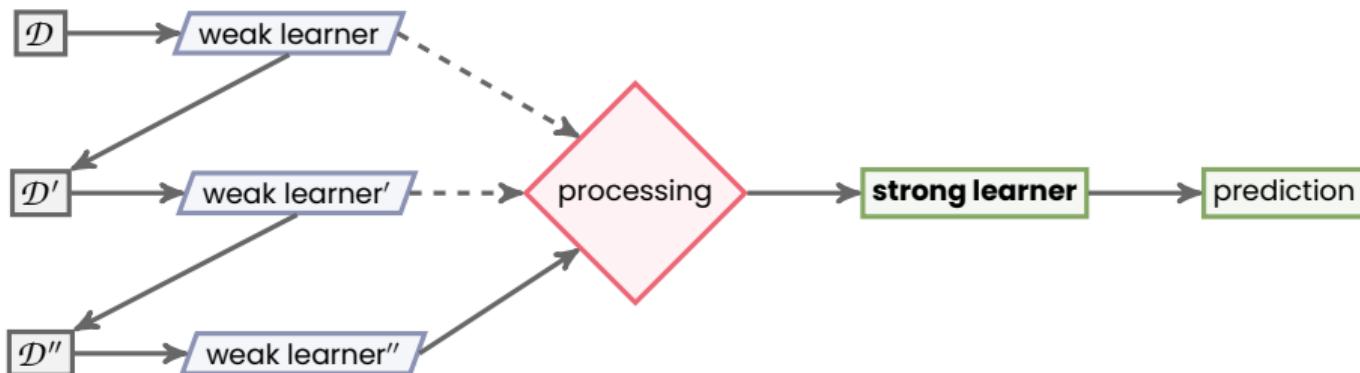
Let $\mathcal{D} = \{(\vec{x}, y)\}$ be a labelled dataset, then a model f is a **strong learner** if

$$\forall \varepsilon > 0 \quad P(f(\vec{x}) \neq y) \leq \varepsilon.$$

Weak Learner

Let $\mathcal{D} = \{(\vec{x}, y)\}$ be a labelled dataset, then a model f is a **weak learner** if

$$\exists \varepsilon' > 0 \quad P(f(\vec{x}) \neq y) \leq \varepsilon'.$$

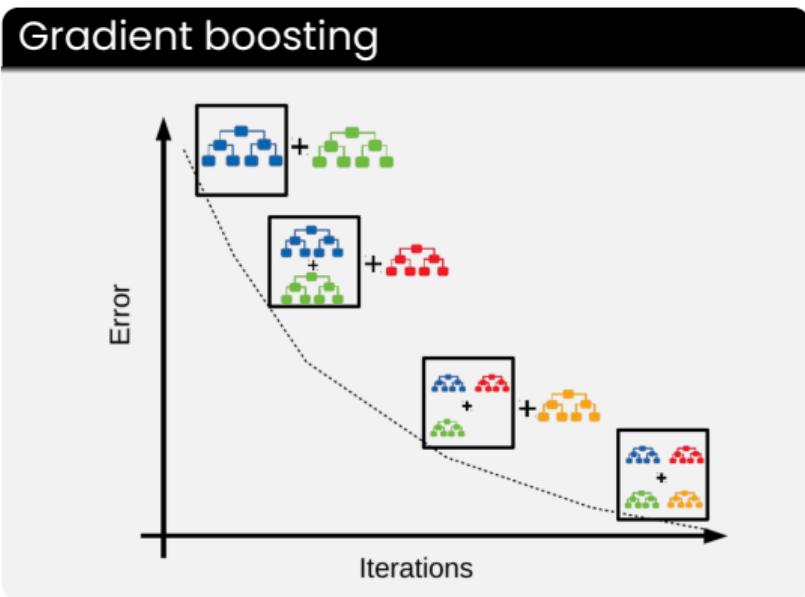
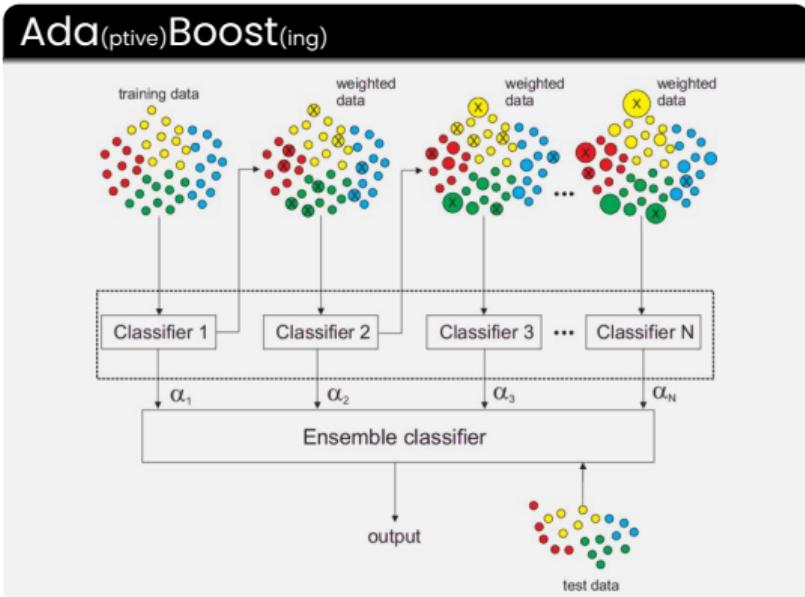




Ensemble Learning

Boosting algorithms

Let $\mathcal{D} = \{(\vec{x}_i, y_i) \mid y_i \in \{-1, 1\} \forall i = 1, 2, \dots, N\}$, and \mathcal{H} be a *weak learner* on \mathcal{D} :





Ensemble Learning

Boosting algorithms

Let $\mathcal{D} = \{(\vec{x}_i, y_i) \mid y_i \in \{-1, 1\} \forall i = 1, 2, \dots, N\}$, and \mathcal{H} be a *weak learner* on \mathcal{D} :

Ada_(ptive)Boost_(ing)

Idea: weighted majority voting

Require: $M > 0$

Require: $m = 0, w_i^{(0)} \leftarrow N^{-1}, \forall i = 1, 2, \dots, N$

for $0 < m \leq M$ **do**

run $\mathcal{H}^{(m)}$ on \mathcal{D}
 $\beta^{(m)} \leftarrow \sum_{i=1}^N w_i^{(m)} \theta(-y_i \mathcal{H}^{(m)}(\vec{x}_i))$ \triangleright error rate

$\alpha^{(m)} = \frac{1}{2} \ln \frac{1 - \beta^{(m)}}{\beta^{(m)}}$

$w_i^{(m+1)} \leftarrow w_i^{(m)} \text{softmax}_{\alpha^{(m)}}(y_i \mathcal{H}^{(m)}(\vec{x}_i))$

return $\mathcal{H}(\vec{x}) = \text{sign} \left(\sum_{m=1}^M \alpha^{(m)} \mathcal{H}^{(m)}(\vec{x}) \right)$.

Remember that θ is the Heaviside function: $\theta(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$, and $\text{softmax}_\beta(z) = \frac{\exp(-\beta z)}{\sum_{i=1}^N \exp(-\beta z_i)}$.

Gradient boosting

Idea: improving on previous attempts

Require: $M > 0, m = 0, \nu > 0, \mathcal{H}(\vec{x}) = \gamma + \sum_{i=1}^M \gamma^{(m)} h^{(m)}(\vec{x}), h^{(m)}$

weak learner, $\mathcal{H}^{(0)}(\vec{x}) = \arg \min_{\gamma} \mathcal{L}(y, \gamma)$

Require: $\mathcal{H}^{(m)} = \mathcal{H}^{(m-1)} + \arg \min_{\gamma^{(m)}, h^{(m)}} \mathcal{L}(y, \mathcal{H}^{(m-1)}(\vec{x}) +$

$\gamma^{(m)} h^{(m)}(\vec{x}))$

for $0 < m \leq M$ **do**

$r_i^{(m)} = -\frac{\delta \mathcal{L}}{\delta \mathcal{H}} \Big|_{\mathcal{H}=\mathcal{H}^{(m)}} \text{ for } i = 1, 2, \dots, N$

train $\mathcal{H}^{(m)}$ on $\{(\vec{x}, r^{(m)})\}$

$\gamma^{(m)} \leftarrow \arg \min_y \mathcal{L}(y, \mathcal{H}^{(m-1)}(\vec{x}) + \nu \gamma^{(m)} h^{(m)}(\vec{x}))$

$\mathcal{H}^{(m)} \leftarrow \mathcal{H}^{(m-1)} + \nu \gamma^{(m)} h^{(m)}$

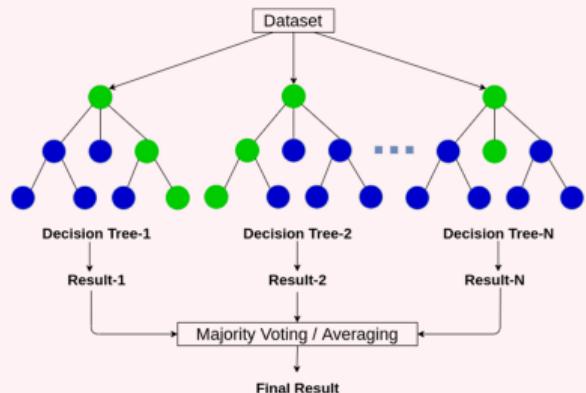
return $\mathcal{H}^{(M)}(\vec{x})$. $\triangleright \nu$ learning rate



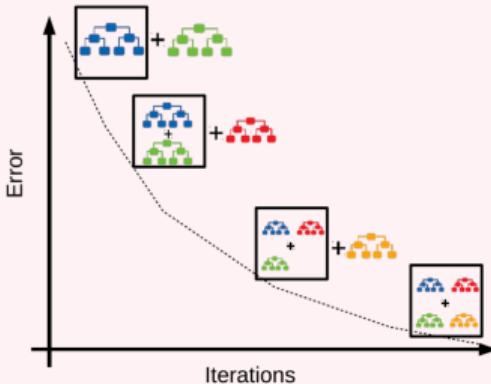
Ensemble Learning

Random forests and boosted decision trees

??? | Random Forest



??? | Boosted Decision Trees



Remember that for $Y = (y_1, \dots, y_B)$ i.i.d. (variance σ^2 and pairwise correlation ρ):

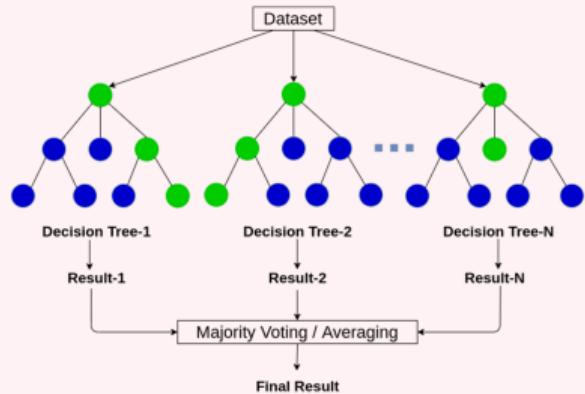
$$\rho = \frac{\text{Cov}(Y)}{\sigma^2} \Leftrightarrow \text{Cov}(\bar{Y}) = \rho\sigma^2 + \frac{1 - \rho}{B}\sigma^2$$



Ensemble Learning

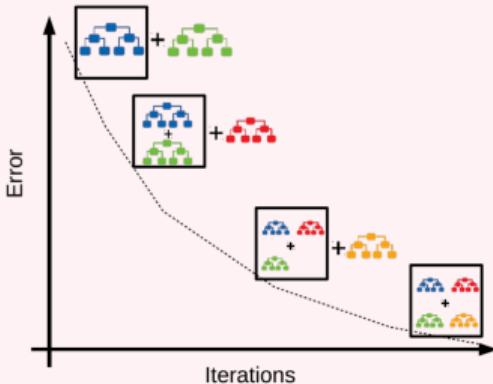
Random forests and boosted decision trees

??? | Random Forest



Variance → ?
Bias → ?

??? | Boosted Decision Trees



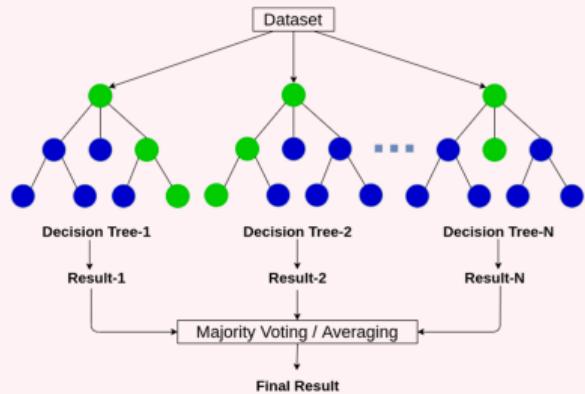
Variance → ?
Bias → ?



Ensemble Learning

Random forests and boosted decision trees

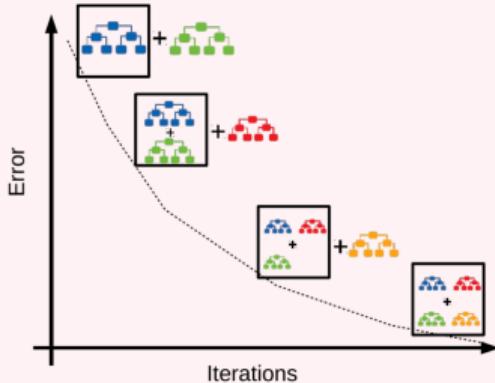
Bagging | Random Forest



Variance → reduction ($\text{Cov}(\bar{Y}) \xrightarrow{B \gg 1} \rho\sigma^2 \leq \sigma^2$)
Bias → increase (more restrictions)

Trees in **random forests** are usually **fully-grown** to start with a low bias, and to reduce bias after bagging.

Boosting | Boosted Decision Trees



Variance → increase
Bias → decrease

Trees in **gradient boosting** are usually **shallow** to start with high bias, and decrease it after boosting.



4 Neural Networks



Computational graphs

Table of contents

1. Some History and Philosophy to Start

2. The ML Mindset

3. ML Algorithms

4. Neural Networks

Computational graphs

Non Linearity of Neural Networks

Approximation theorems

Neural network training

Regularisation of neural networks

5. Conclusions





Computational Graphs

Preliminaries

x_0

x_1

x_2

x_3

x_4

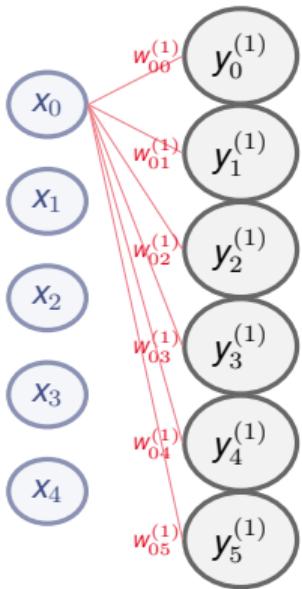
Let $f^{(n)}$ be one of N affine functions:

$$f^{(n)} : \mathbb{R}^{W_{(n-1)}} \rightarrow \mathbb{R}^{W_n}, \quad n = 1, 2, \dots, N$$



Computational Graphs

Preliminaries



Let $f^{(n)}$ be one of N affine functions:

$$f^{(n)} : \mathbb{R}^{w_{(n-1)}} \rightarrow \mathbb{R}^{w_{(n)}}, \quad n = 1, 2, \dots, N$$

s.t.

$$\vec{y}^{(n)} = f^{(n)} \left(\vec{y}^{(n-1)} \right) = W^{(n)} \vec{y}^{(n-1)} + \vec{b}^{(n)},$$

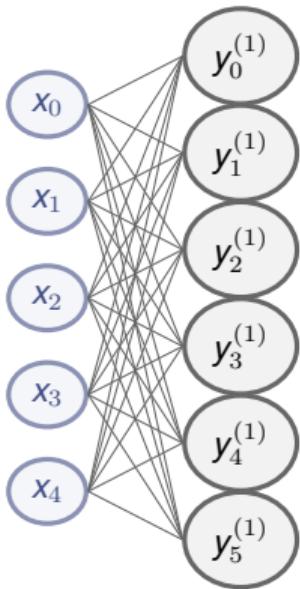
and $\vec{y}^{(0)} = \vec{x}$ (w : "weights", b : "bias"):

$$\vec{y}^{(N)} = f^{(N)} \circ f^{(N-1)} \circ \dots \circ f^{(1)}(\vec{x})$$



Computational Graphs

Preliminaries



Let $f^{(n)}$ be one of N affine functions:

$$f^{(n)} : \mathbb{R}^{w_{(n-1)}} \rightarrow \mathbb{R}^{w_n}, \quad n = 1, 2, \dots, N$$

s.t.

$$\vec{y}^{(n)} = f^{(n)} \left(\vec{y}^{(n-1)} \right) = W^{(n)} \vec{y}^{(n-1)} + \vec{b}^{(n)},$$

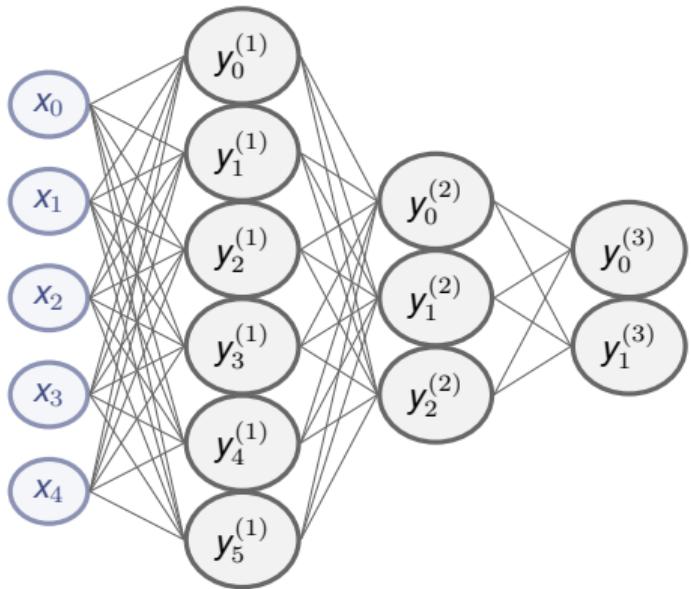
and $\vec{y}^{(0)} = \vec{x}$ (w : "weights", b : "bias"):

$$y^{(N)} = f^{(N)} \circ f^{(N-1)} \circ \dots \circ f^{(1)}(\vec{x})$$



Computational Graphs

Preliminaries



Let $f^{(n)}$ be one of N affine functions:

$$f^{(n)} : \mathbb{R}^{W_{(n-1)}} \rightarrow \mathbb{R}^{W_n}, \quad n = 1, 2, \dots, N$$

s.t.

$$\vec{y}^{(n)} = f^{(n)} \left(\vec{y}^{(n-1)} \right) = W^{(n)} \vec{y}^{(n-1)} + \vec{b}^{(n)},$$

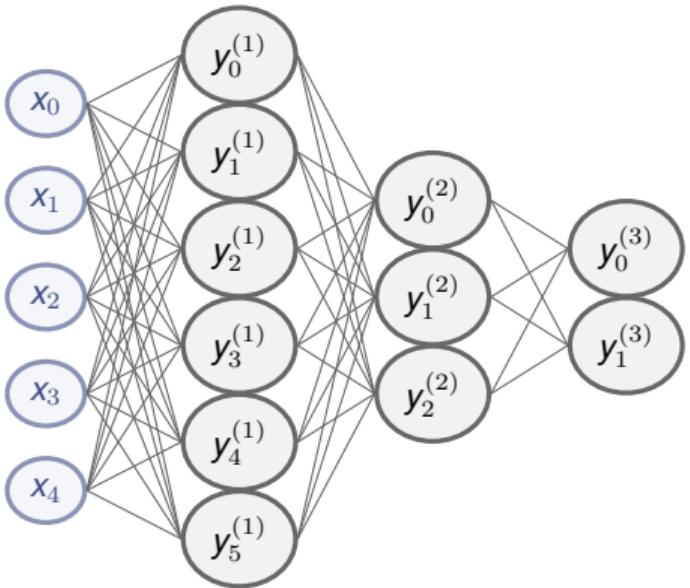
and $\vec{y}^{(0)} = \vec{x}$ (W : "weights", b : "bias"):

$$\vec{y}^{(N)} = f^{(N)} \circ f^{(N-1)} \circ \dots \circ f^{(1)}(\vec{x})$$



Computational Graphs

Linearity vs non linearity



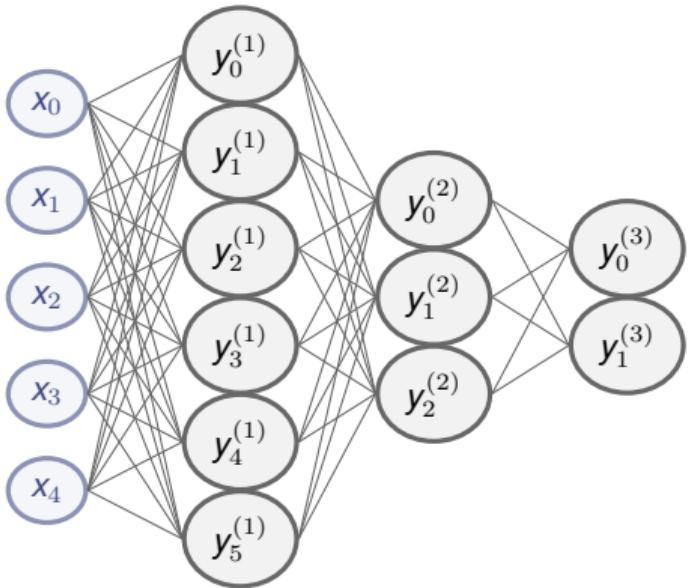
Linearity of the network

$$\vec{y}^{(N)} = \underbrace{W^{(N)} W^{(N-1)} \dots W^{(1)}}_W \vec{x} + \underbrace{\vec{b}^{(N)} + W^{(N)} \vec{b}^{(N-1)} + \dots}_\vec{b}$$



Computational Graphs

Linearity vs non linearity



Linearity of the network

$$\vec{y}^{(N)} = \underbrace{W^{(N)} W^{(N-1)} \dots W^{(1)}}_W \vec{x} + \underbrace{\vec{b}^{(N)} + W^{(N)} \vec{b}^{(N-1)} + \dots}_\vec{b}$$

Activation functions

Let $a^{(n)} : \mathbb{R} \times \mathbb{R}$ **non linear**:

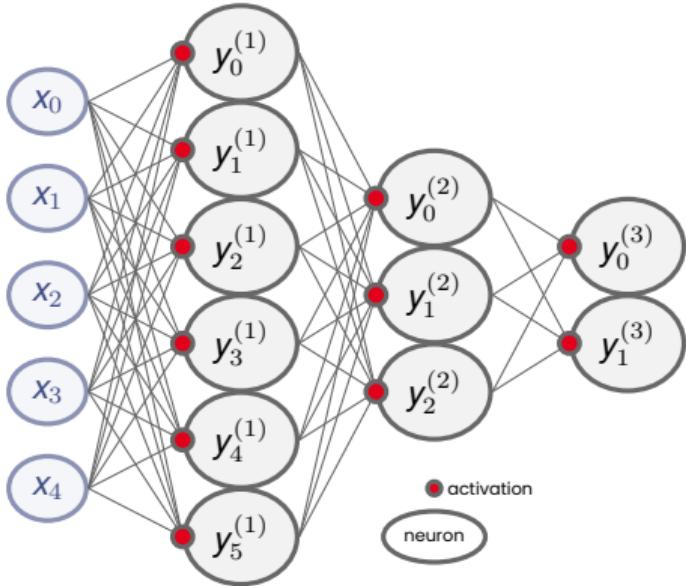
$$\left(g^{(n)}(\vec{x}) \right)_{ij} = a^{(n)} \left(\left(f^{(n)}(\vec{x}) \right)_{ij} \right),$$

where $i = 1, 2, \dots, w_{(n)}$ and $j = 1, 2, \dots, w_{(n-1)}$.



Computational Graphs

Neural networks



The activation on the last layer strongly depends on the task...More on this later!

Call (\odot is the Hadamard product)

$$g^{(n)}(\vec{y}^{(n-1)}) = a^{(n)} \odot (W^{(n)} \vec{y}^{(n-1)} + \vec{b}^{(n)})$$

the n-th **layer** in the graph.

Neural network

The non linear function:

$$g^{(N)} \circ g^{(N-1)} \circ \dots \circ g^{(1)}$$

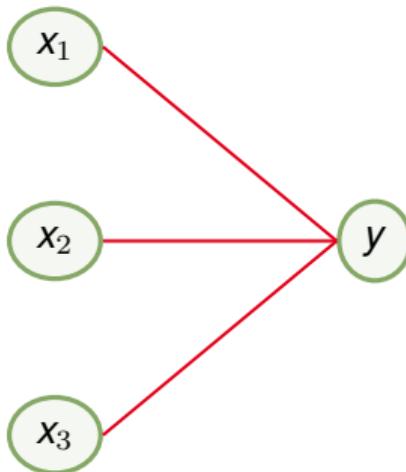
is called a (*fully connected*) **neural network** (NN) with $N - 1$ **hidden layers**.



The (Multi-Layered) Perceptron

The historical context

The structure



is called **perceptron** Rosenblatt (1958) and it represents the fundamental unit of a NN.
A stack of perceptrons is called **Multi-Layered Perceptron** (MLP.)



4 Neural Networks



Non Linearity of Neural Networks

Table of contents

1. Some History and Philosophy to Start

2. The ML Mindset

3. ML Algorithms

4. Neural Networks

Computational graphs

Non Linearity of Neural Networks

Approximation theorems

Neural network training

Regularisation of neural networks

5. Conclusions



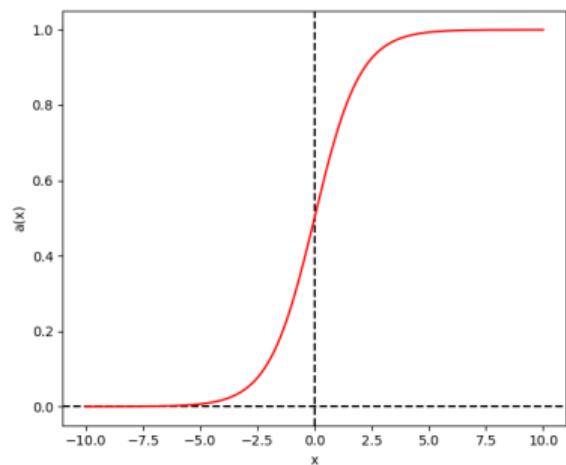


Neural Networks

Activation functions

Activation functions might depend on the task, to ease the training. For instance:

Sigmoid



$$a(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$$

- classically the first...
- gradients might saturate for $x \rightarrow \pm\infty$ see later
- good interpretation as GLM (probability)

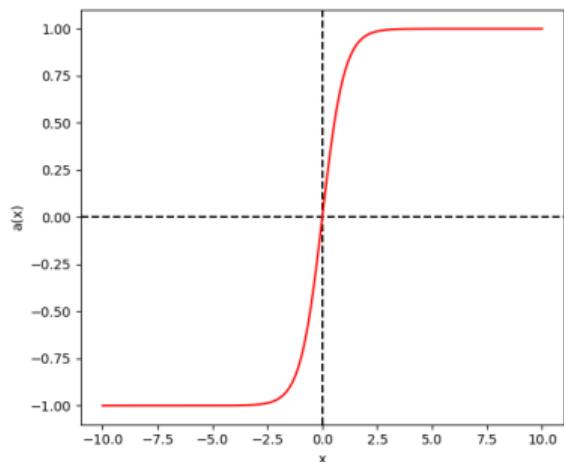


Neural Networks

Activation functions

Activation functions might depend on the task, to ease the training. For instance:

Hyperbolic Tangent



$$a(x) = \tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$

- outputs naturally centred
- might saturate for $x \rightarrow \pm\infty$
- good alternative to σ
- traditionally in (old) GANs

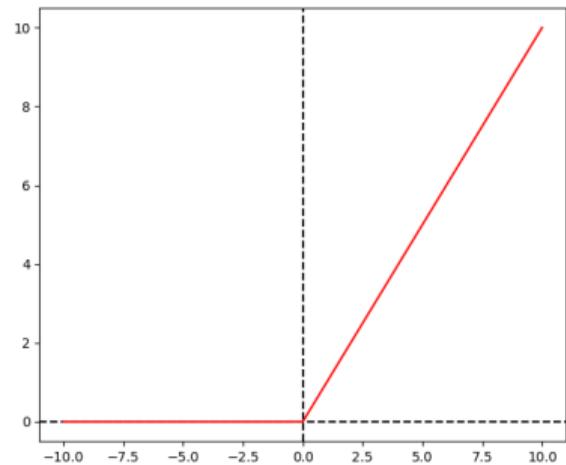


Neural Networks

Activation functions

Activation functions might depend on the task, to ease the training. For instance:

REctified Linear Unit



$$a(x) = \text{ReLU}(x) = \max(0, x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases}$$

- **omnipresent** powerful sparsifier see Glorot et al. (2011)
- gradients might saturate for $x \rightarrow -\infty$
- forces positive outputs (!) q: is it good for output layer?
- slightly non differentiable
- computationally fast

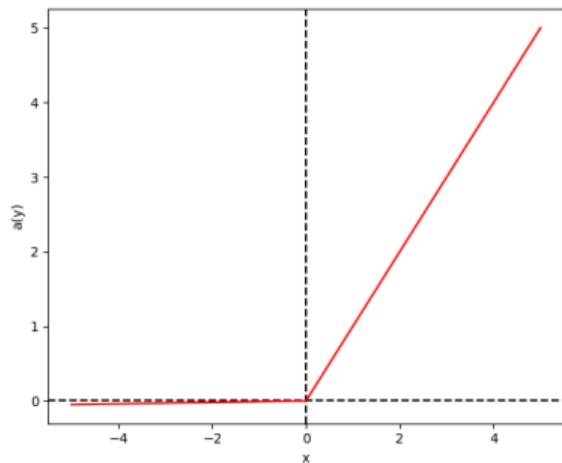


Neural Networks

Activation functions

Activation functions might depend on the task, to ease the training. For instance:

Leaky REctified Linear Unit



$$a(x) = \text{LeakyReLU}_\alpha(x) = \begin{cases} x, & \text{if } x \geq 0 \\ \alpha x, & \text{if } x < 0 \end{cases}$$

- new *slope* hyperparameter $\alpha \in \mathbb{R}^+$
- solves the saturation problem
- negative outputs are slightly allowed
- slightly non differentiable
- computationally fast

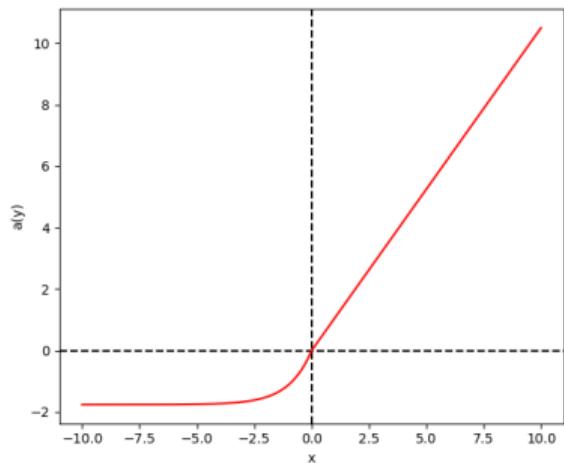


Neural Networks

Activation functions

Activation functions might depend on the task, to ease the training. For instance:

Scaled Exponential Linear Unit



$$\begin{aligned}a(x) &= \text{SELU}(x) \\&= \gamma (\max(0, x) + \min(0, \alpha(e^x - 1)))\end{aligned}$$

- improve NN behaviour see Klambauer et al. (2017)
- solves the saturation problem
- negative outputs are not sparsified
- slightly non differentiable
- requires good initialisation see later...

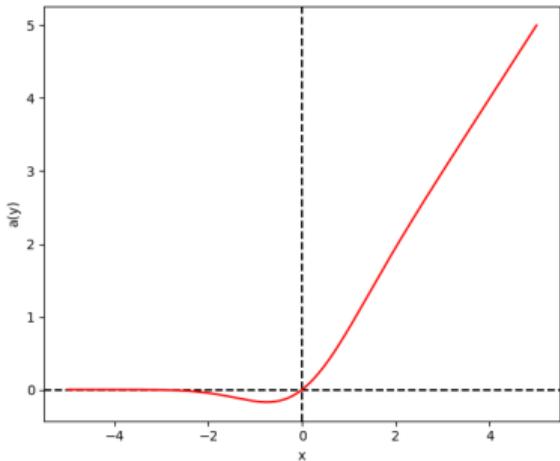


Neural Networks

Activation functions

Activation functions might depend on the task, to ease the training. For instance:

Gaussian Error Linear Unit



$$a(x) = \text{GELU}(x) = x \Phi(x)$$

- stochastic regularisation method see [Hendrycks and Gipel \(2016\)](#)
- might saturate at $x \rightarrow -\infty$, but discouraged
- good normalisation of the activations

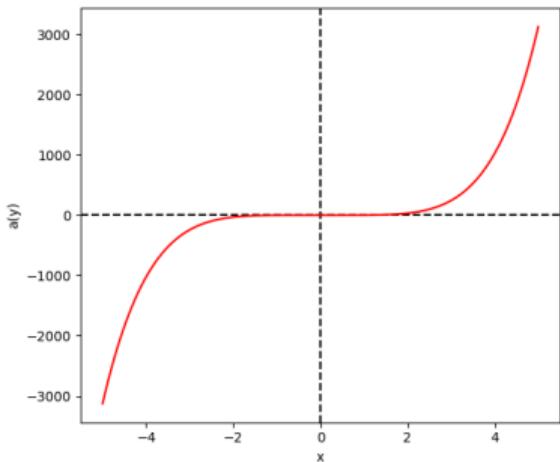


Neural Networks

Activation functions

Activation functions might depend on the task, to ease the training. For instance:

Homogeneous activation



$$a(x) = x^p, \quad | \quad a(\lambda x) = \lambda^p a(x)$$

- good behaviour of the network
- might help approximations
- useful in scientific/“physics informed” scenarios
- use at your own risk...

“We are all responsible users” (from the [Python guide](#))



Neural Networks

Activation functions

Activation functions might depend on the task, to ease the training. For instance:

Last layer activations

Binary classification

$$a^{(N)}(\vec{x})_i = \sigma(x_i) \\ = \frac{1}{1 + e^{-x_i}} \in [0, 1]$$

Multiclass classification

$$a^{(N)}(\vec{x})_i = \text{softmax}(x_i) \\ = \frac{e^{x_i}}{\sum_{i=1}^K e^{x_i}} \in [0, 1]^K$$

Regression

$$a^{(N)}(\vec{x})_i = \text{Id}(x_i)$$

Activations are quite flexible and strongly depend on the type of task required (e.g.: if all outputs are positives, ReLU might be used for a regression task)!



4 Neural Networks

- Approximation theorems

Table of contents

1. Some History and Philosophy to Start

2. The ML Mindset

3. ML Algorithms

4. Neural Networks

Computational graphs

Non Linearity of Neural Networks

Approximation theorems

Neural network training

Regularisation of neural networks

5. Conclusions



Riccardo Finotello

AIPhy

30/09/2024

83 / 102



Neural Networks

Universal approximation theorems

Theorem (Cybenko | Approximation by sigmoid-activated NNs)

Let \mathcal{F} be the set of $\mathcal{C}^1([0, 1]^n)$ scalar functions, and σ be a **sigmoid** function. Then

$$\exists N > 0 \mid f(\vec{x}) = \sum_{i=1}^N \alpha_i \sigma(\vec{w}_i \cdot \vec{x} + b)$$

is dense in \mathcal{F} .

In simple words: using a 1-layer deep sigmoid-activated scalar NN we can approximate with arbitrary precision any $\mathcal{C}([0, 1])$ scalar function. Let $g(\vec{x}) \in \mathcal{C}([0, 1])$ and $f(\vec{x})$ be such NN, then

$$\forall \varepsilon > 0, \quad \sup_{\vec{x}} ||f(\vec{x}) - g(\vec{x})|| < \varepsilon.$$



Neural Networks

Universal approximation theorems

Theorem (Kolmogorov–Arnold | Approximation theorem)

Let f be a function in \mathcal{F} , then

$$f(\vec{x}) = \sum_{q=0}^{2n} \phi_q \left(\sum_{p=1}^n \varphi_{q,p}(x_p), \right)$$

where ϕ and φ are continuous scalar functions of a single variable.

In simple words: the only “needed” functions are single-variable activations and sums. Technically, if we could choose the activations of each unit (neuron), we could exactly write any multivariate function as superposition of univariate functions.

might be interesting to some of you: [Liu et al. \(2024\)](#)



Neural Networks

Universal approximation theorems

Theorem (Width expressivity of NNS see [Lu et al. \(2017\)](#))

Let $g: \mathbb{R}^n \rightarrow \mathbb{R}$ be a Lebesgue integrable function, and \mathfrak{F} be the set of **fully connected ReLU-activated NN** with width $w \leq n + 4$:

$$\exists f \in \mathfrak{F} \mid \forall \varepsilon > 0 \int_{\mathbb{R}^n} |g(\vec{x}) - f(\vec{x})| < \varepsilon$$

In other words: width-bounded NNs can be used as universal approximators on the entire domain of definition. It is also curious to see:

$$w \leq n \Rightarrow \int_{\mathbb{R}^n} |g(\vec{x}) - f(\vec{x})| \text{ diverges,}$$

hence the restrictions to $[-1, 1]^n$ (i.e. good normalisation):

$$w \leq n - 1 \Rightarrow \exists \varepsilon' > 0 \mid \int_{[-1,1]^n} |g(\vec{x}) - f(\vec{x})| \geq \varepsilon'.$$



Neural Networks

Universal approximation theorems

Theorem (Trade-off width/depth see Lu et al. (2017))

Let n be the input dimensions. For any integer $k \geq n + 4$, there exists a ReLU-activated NN $f: \mathbb{R}^n \rightarrow \mathbb{R}$ with width $w = 2k^2$ and depth $d = 3$, such that

$$\forall b > 0, \forall g: \mathbb{R}^n \rightarrow \mathbb{R},$$

where g is a ReLU-activated NN whose parameters are bounded in $[-b, b]$, with width $w' \leq k^{\frac{3}{2}}$ and depth $d \leq k + 2$, it is true that

$$\exists \varepsilon > 0 \mid \int_{\mathbb{R}^n} (f(\vec{x}) - g(\vec{x}))^2 \geq \varepsilon.$$

In other words: any decrease in width, should be compensated by an increase in depth to keep the same expressivity of the NNs.



Neural Networks

Universal approximation theorems

Some needed questions

Q1: fully connected NNs are universal approximators! Did we answer *the ultimate question of life, the universe and everything?*



Neural Networks

Universal approximation theorems

Some needed questions

Q1: fully connected NNs are universal approximators! Did we answer *the ultimate question of life, the universe and everything?*

- the answer is “42”, not “neural networks”... 🤪
- **existence** of sth $\not\Rightarrow$ easy to find
- fully-connected NNs **low bias** $\not\Rightarrow$ not all kinds of inputs are adapted



Neural Networks

Universal approximation theorems

Some needed questions

Q1: fully connected NNs are universal approximators! Did we answer *the ultimate question of life, the universe and everything?*

- the answer is “42”, not “neural networks”... 🤪
- **existence** of sth $\not\Rightarrow$ easy to find
- fully-connected NNs **low bias** $\not\Rightarrow$ not all kinds of inputs are adapted

Q2: 🤦 shut up, I don't care! Suppose we found a perfect NN: can we deploy it for the world to see and use?



Neural Networks

Universal approximation theorems

Some needed questions

Q1: fully connected NNs are universal approximators! Did we answer *the ultimate question of life, the universe and everything?*

- the answer is “42”, not “neural networks”... 🤪
- **existence** of sth $\not\Rightarrow$ easy to find
- fully-connected NNs **low bias** $\not\Rightarrow$ not all kinds of inputs are adapted

Q2: 🤦 shut up, I don’t care! Suppose we found a perfect NN: can we deploy it for the world to see and use?

- you would lead us to another AI winter...
- NNs are trained on **samples** \Rightarrow predict conditioned on that (+ some extrapolation)
- probably ok with **infinite** amount of data (**population**), but how to train?



4 Neural Networks



Neural network training

Table of contents

1. Some History and Philosophy to Start

2. The ML Mindset

3. ML Algorithms

4. Neural Networks

Computational graphs

Non Linearity of Neural Networks

Approximation theorems

Neural network training

Regularisation of neural networks

5. Conclusions



Riccardo Finotello

AIPhy

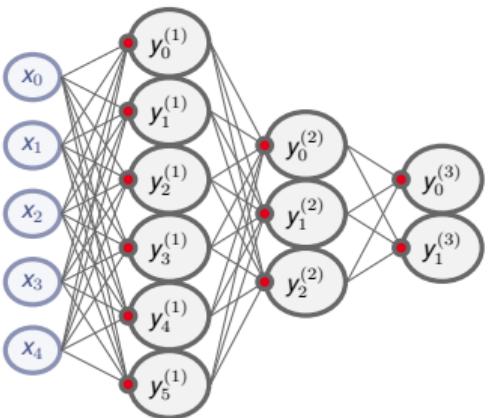
30/09/2024

86 / 102



Neural Network Training

Backpropagation

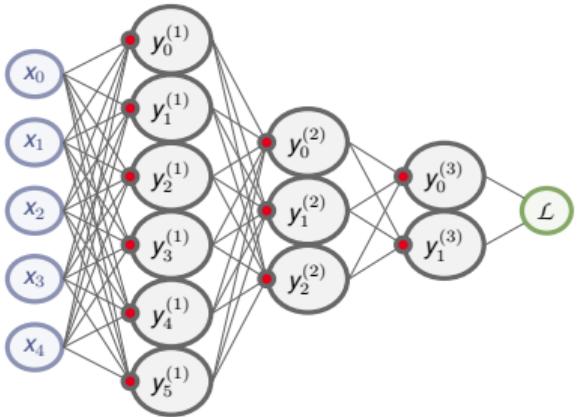


see Rumelhart et al. (1986)



Neural Network Training

Backpropagation



see Rumelhart et al. (1986)

Let $\mathcal{L} : \mathbb{R}^{w(N)} \rightarrow \mathbb{R}$ be a **loss function** (it depends on the task) and append it to the NN, where at the ℓ -th layer:

$$y_i^{(\ell+1)} = a^{(\ell)}(z_i^{(\ell)}),$$
$$z_i^{(\ell)} = \sum_{j=1}^{w^{(\ell-1)}} W_{ij}^{(\ell)} y_j^{(\ell-1)} + b_i^{(\ell)}$$

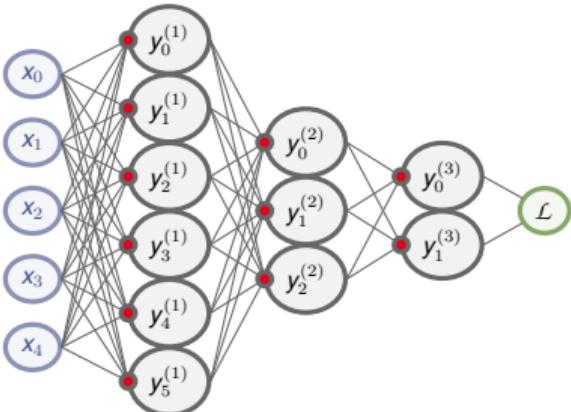


Neural Network Training

Backpropagation

see Rumelhart et al. (1986)

Let $\mathcal{L} : \mathbb{R}^{W(N)} \rightarrow \mathbb{R}$ be a **loss function** (it depends on the task) and append it to the NN, where at the ℓ -th layer:



$$y_i^{(\ell+1)} = a^{(\ell)}(z_i^{(\ell)}),$$

$$z_i^{(\ell)} = \sum_{j=1}^{w^{(\ell-1)}} W_{ij}^{(\ell)} y_j^{(\ell-1)} + b_i^{(\ell)}$$

We can perform **gradient descent** (GD) for each $W^{(\ell)}, \ell = 1, 2, \dots, N$ by computing:

$$\begin{cases} \frac{\partial \mathcal{L}}{\partial W_{ij}^{(\ell)}} &= \sum_{k=1}^{w^{(\ell)}} \frac{\partial \mathcal{L}}{\partial z_k^{(\ell)}} \frac{\partial z_k^{(\ell)}}{\partial W_{ij}^{(\ell)}} = \frac{\partial \mathcal{L}}{\partial z_i^{(\ell)}} y_j^{(\ell-1)} \\ \frac{\partial \mathcal{L}}{\partial b_i^{(\ell)}} &= \sum_{k=1}^{w^{(\ell)}} \frac{\partial \mathcal{L}}{\partial z_k^{(\ell)}} \frac{\partial z_k^{(\ell)}}{\partial b_i^{(\ell)}} = \frac{\partial \mathcal{L}}{\partial z_i^{(\ell)}} \end{cases}$$

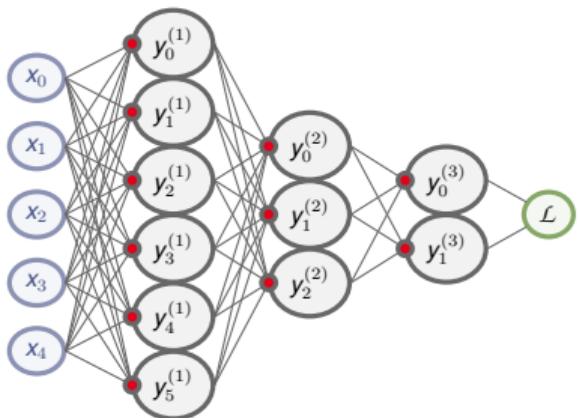


Neural Network Training

Backpropagation

see Rumelhart et al. (1986)

From the previous expression:



$$\delta_i^{(\ell-1)} = \frac{\partial \mathcal{L}}{\partial z_i^{(\ell-1)}}$$

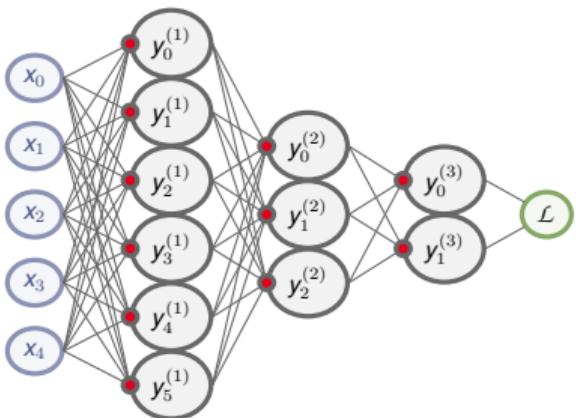


Neural Network Training

Backpropagation

see Rumelhart et al. (1986)

From the previous expression:



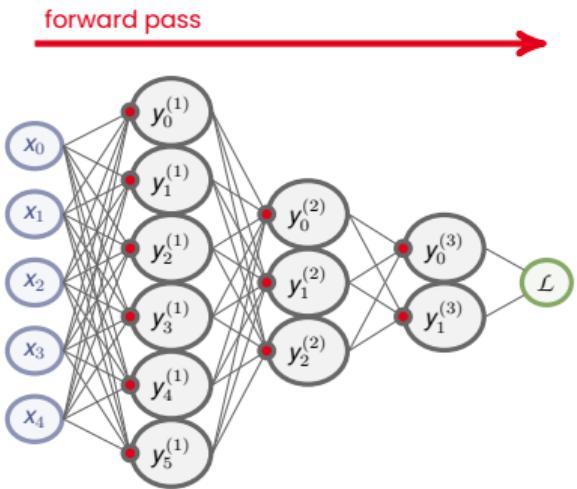
$$\begin{aligned}\delta_i^{(\ell-1)} &= \frac{\partial \mathcal{L}}{\partial z_i^{(\ell-1)}} \\ &= \sum_{k=1}^{w^{(\ell)}} \frac{\partial \mathcal{L}}{\partial z_k^{(\ell)}} \frac{\partial z_k^{(\ell)}}{\partial z_i^{(\ell-1)}} \\ &= \sum_{k=1}^{w^{(\ell)}} \sum_{h=1}^{w^{(\ell-1)}} \delta_k^{(\ell)} \frac{\partial z_k^{(\ell)}}{\partial a_h^{(\ell-1)}} \frac{\partial a_h^{(\ell-1)}}{\partial z_i^{(\ell-1)}} \\ &= \left(a^{(\ell-1)} \right)' \sum_{k=1}^{w^{(\ell)}} \delta_k^{(\ell)} W_{ki}^{(\ell)}\end{aligned}$$



Neural Network Training

Backpropagation

see Rumelhart et al. (1986)



Training a NN is a **two steps** procedure:

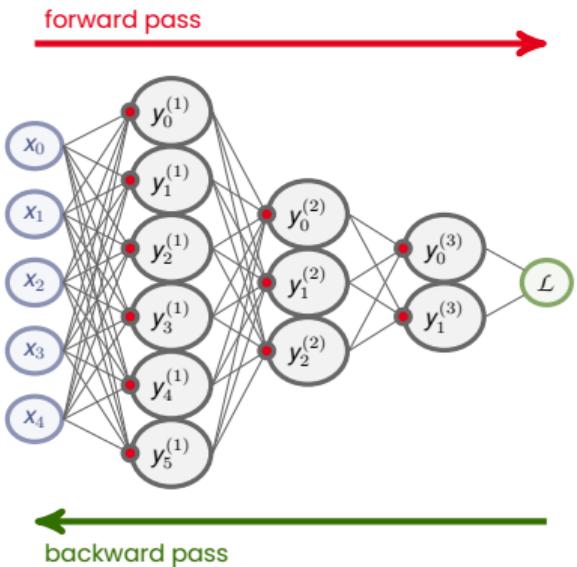
1. during the **forward pass** the outputs and outputs of each layer (loss included) are computed and **stored**



Neural Network Training

Backpropagation

see Rumelhart et al. (1986)



Training a NN is a **two steps** procedure:

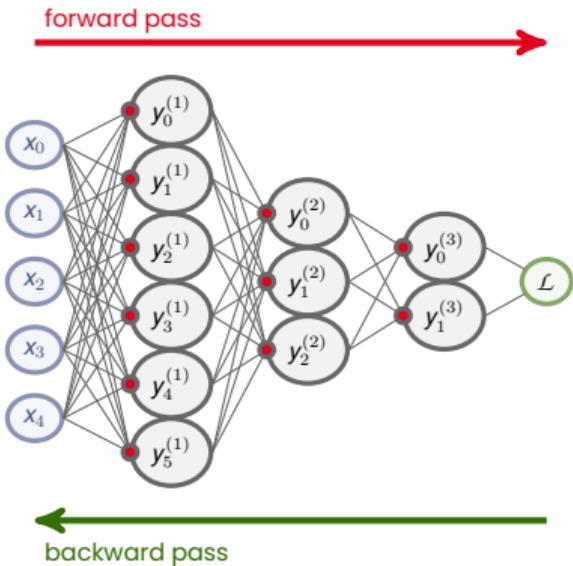
1. during the **forward pass** the outputs and outputs of each layer (loss included) are computed and **stored**
2. in the **backward pass** the gradients of each layer are assembled **iteratively**



Neural Network Training

Backpropagation

see Rumelhart et al. (1986)



Training a NN is a **two steps** procedure:

1. during the **forward pass** the outputs and outputs of each layer (loss included) are computed and **stored**
2. in the **backward pass** the gradients of each layer are assembled **iteratively**

Finally, the **update** of the parameters is:

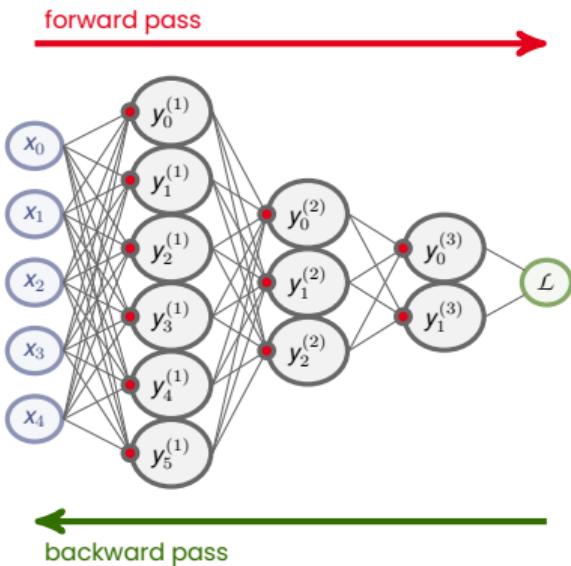
$$\begin{cases} W_{ij}^{(\ell)} & \leftarrow W_{ij}^{(\ell)} - \alpha \frac{\partial \mathcal{L}}{\partial W_{ij}^{(\ell)}} \\ b_i^{(\ell)} & \leftarrow b_i^{(\ell)} - \alpha \frac{\partial \mathcal{L}}{\partial b_i^{(\ell)}} \end{cases},$$

where α is the *learning rate hyperparameter*.



Neural Network Training

Backpropagation



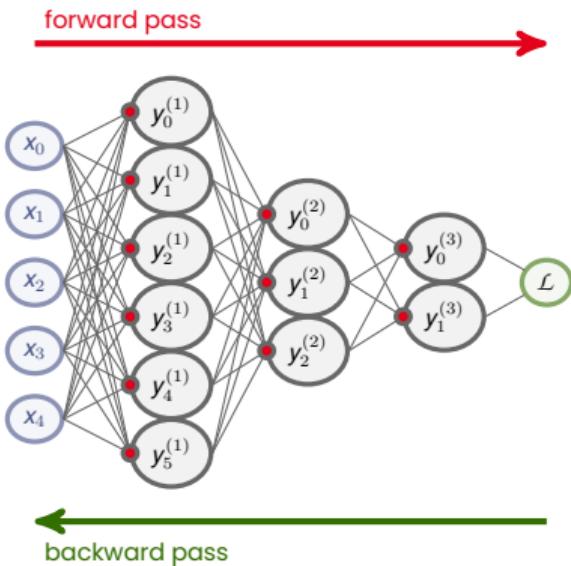
Q: what are good **initialisations** of weights and biases?

- NNs **propagate** by matrix multiplication
- gradients **large** to update
- gradients **small** not to explode



Neural Network Training

Backpropagation



Q: what are good **initialisations** of weights and biases?

- NNs **propagate** by matrix multiplication
- gradients **large** to update
- gradients **small** not to explode

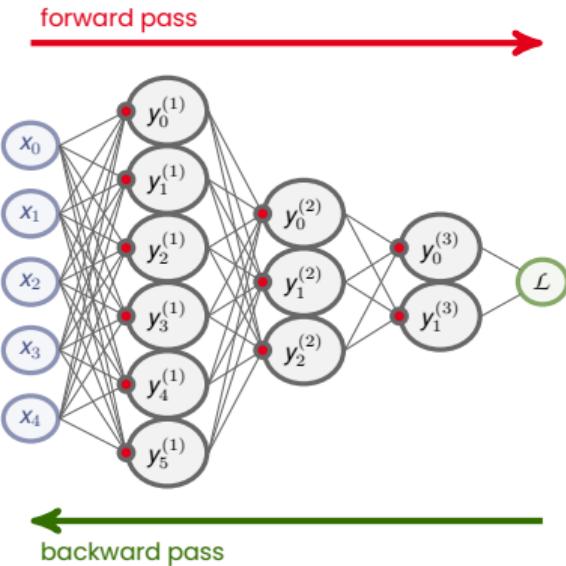
However, we know:

$$\begin{cases} \frac{\partial \mathcal{L}}{\partial W_{ij}^{(\ell)}} &= \delta_i^{(\ell)} y_j^{(\ell-1)} \\ \frac{\partial \mathcal{L}}{\partial b_i^{(\ell)}} &= \delta_i^{(\ell)} \end{cases}$$

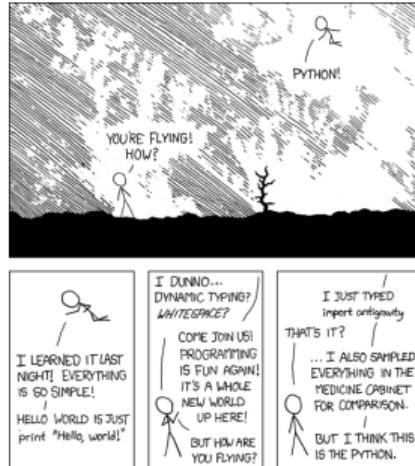


Neural Network Training

Backpropagation



Backpropagation: usually boilerplate code which is already available in most frameworks
(Pytorch, Lightning, Tensorflow, Keras, etc.)



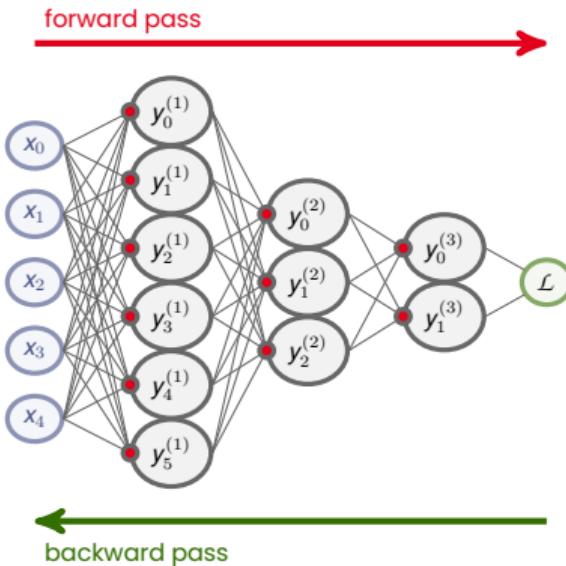
(xkcd.com)





Neural Network Training

Backpropagation



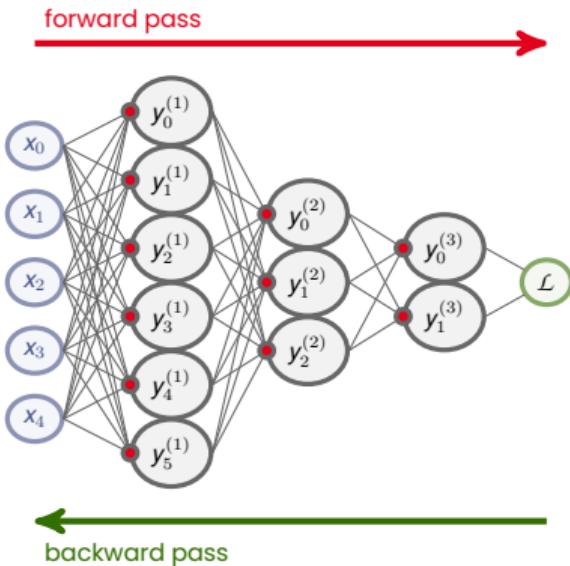
THINK

What would happen if all $W_{ij}^{(\ell)}$ were to be initialised to the same constant (say 0)
 $\forall \ell = 1, 2, \dots, N$?



Neural Network Training

Backpropagation



THINK

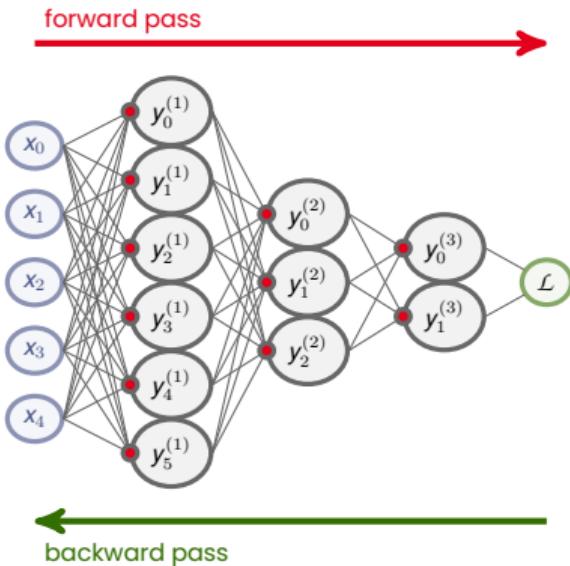
What would happen if all $W_{ij}^{(\ell)}$ were to be initialised to the same constant (say 0)
 $\forall \ell = 1, 2, \dots, N$?

All activations $y^{(\ell)}$ would be the **same!**



Neural Network Training

Backpropagation



THINK

What would happen if all $W_{ij}^{(\ell)}$ were to be initialised to the same constant (say 0)
 $\forall \ell = 1, 2, \dots, N$?

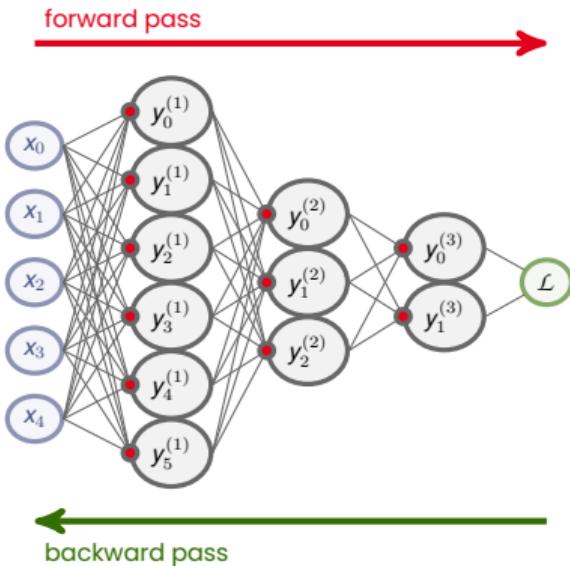
All activations $y^{(\ell)}$ would be the **same!**

What about $\delta^{(\ell)}$?



Neural Network Training

Backpropagation



THINK

What would happen if all $W_{ij}^{(\ell)}$ were to be initialised to the same constant (say 0)
 $\forall \ell = 1, 2, \dots, N$?

All activations $y^{(\ell)}$ would be the **same!**

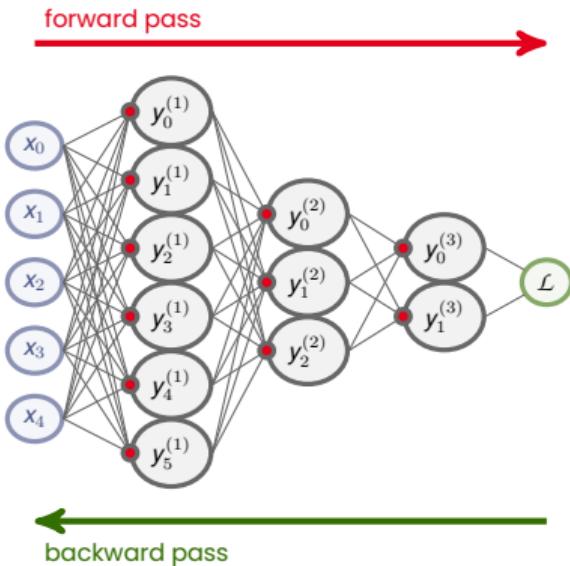
What about $\delta^{(\ell)}$?

All updates $\delta^{(\ell)}$ would be the **same!**



Neural Network Training

Backpropagation



THINK

What would happen if all $W_{ij}^{(\ell)}$ were to be initialised to the same constant (say 0)
 $\forall \ell = 1, 2, \dots, N$?

All activations $y^{(\ell)}$ would be the **same**!

What about $\delta^{(\ell)}$?

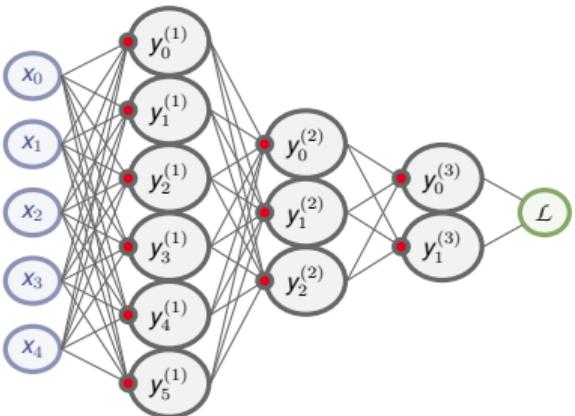
All updates $\delta^{(\ell)}$ would be the **same**!

Nothing to learn!



Neural Network Training

Weight initialisation



Initialisation

It is **fundamental to break the symmetry** (at least for $W^{(\ell)}$):

- initialise with **random** values $\mathcal{N}(\mu, \sigma^2)$
- avoid **large** entries
- follow good **rules of thumb**:

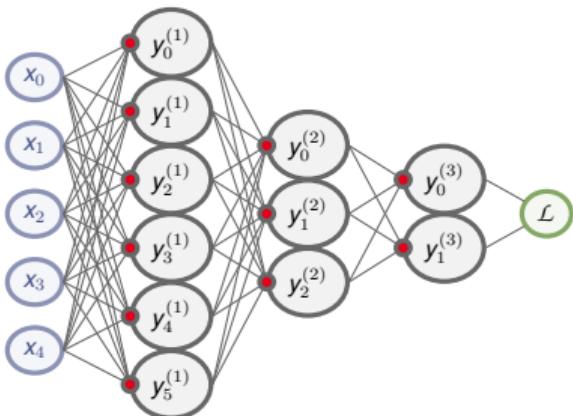
$$\mathbb{E} [y^{(\ell)}] = \mathbb{E} [y^{(\ell-1)}] = 0$$

$$\text{Var}(y^{(\ell)}) = \text{Var}(y^{(\ell-1)})$$



Neural Network Training

Weight initialisation



Some examples

- *LeCun* initialisation ([LeCun et al. \(1998\)](#))

$$W_{ij}^{(\ell)} \sim \mathcal{N} \left(0, \left(w^{(\ell-1)} \right)^{-1} \right), \quad b_i^{(\ell)} = 0$$

for normally centred activations

- *Xavier/Glorot* initialisation ([Glorot and Bengio \(2010\)](#))

$$W_{ij}^{(\ell)} \sim \mathcal{N} \left(0, 2 \left(w^{(\ell)} + w^{(\ell-1)} \right)^{-1} \right) \quad b_i^{(\ell)} = 0$$

for sigmoid/tanh activations

- *(Kaiming) He* initialisation ([He et al. \(2015\)](#))

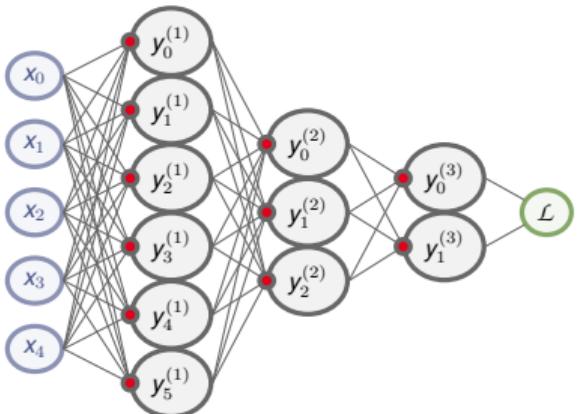
$$W_{ij}^{(\ell)} \sim \mathcal{N} \left(0, 2 \left(w^{(\ell-1)} \right)^{-1} \right) \quad b_i^{(\ell)} = 0$$

for ReLU-family activations



Neural Network Training

Weight initialisation



Some examples

- *LeCun* initialisation ([LeCun et al. \(1998\)](#))

$$W_{ij}^{(\ell)} \sim \mathcal{N} \left(0, \left(w^{(\ell-1)} \right)^{-1} \right), \quad b_i^{(\ell)} = 0$$

- *Xavier/Glorot* initialisation ([Glorot and Bengio \(2010\)](#))

$$W_{ij}^{(\ell)} \sim \mathcal{N} \left(0, 2 \left(w^{(\ell)} + w^{(\ell-1)} \right)^{-1} \right) \quad b_i^{(\ell)} = 0$$

- *(Kaiming) He* initialisation ([He et al. \(2015\)](#))

$$W_{ij}^{(\ell)} \sim \mathcal{N} \left(0, 2 \left(w^{(\ell-1)} \right)^{-1} \right) \quad b_i^{(\ell)} = 0$$

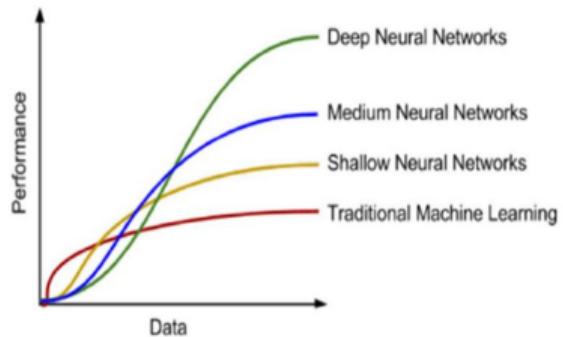
HOMEWORK

- Derive the formula of LeCun initialisation – or look it up, it is still cool!
- Derive the formula of Kaiming He initialisation (what is the difference?)



Neural Network Training

Mini-batch gradient descent

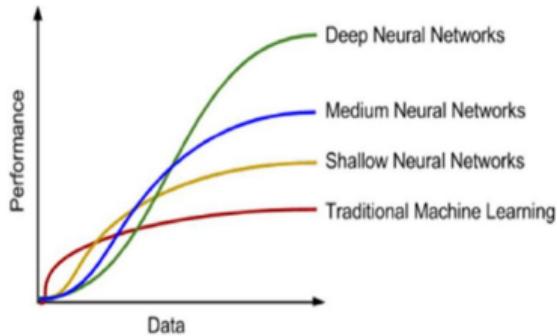


- NNs are powerful at learning/digesting huge amounts of data
- PCs might not be able to load everything all at once
- how to process lots of data?



Neural Network Training

Mini-batch gradient descent



- NNs are powerful at learning/digesting huge amounts of data
- PCs might not be able to load everything all at once
- how to process lots of data?

Require: dataset $\mathcal{D} = \{(\vec{x}, y)\}$

Require: $\{\mathcal{D}_{[b]}\}_{b \in [1, \mathcal{B}]}$ s.t. $\bigcup_{i=0}^{\mathcal{B}-1} \mathcal{D}_{[b]} = \mathcal{D}$

for $0 \leq b < \mathcal{B}$ **do**

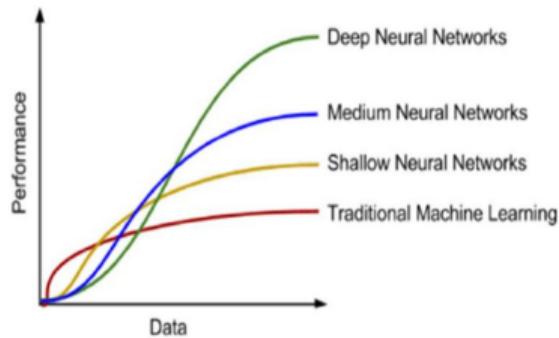
compute forward pass on $\mathcal{D}_{[b]}$
perform *backpropagation*
update $W^{(l)}$ and $b^{(l)}$

return trained NN



Neural Network Training

Mini-batch gradient descent



animation by Luis Medina

- NNs are powerful at learning/digesting huge amounts of data
- PCs might not be able to load everything all at once
- how to process lots of data?

Require: dataset $\mathcal{D} = \{(\vec{x}, y)\}$

Require: $\{\mathcal{D}_{[b]}\}_{b \in [1, \mathcal{B}]}$ s.t. $\bigcup_{i=0}^{\mathcal{B}-1} \mathcal{D}_{[b]} = \mathcal{D}$

for $0 \leq b < \mathcal{B}$ **do**

compute forward pass on $\mathcal{D}_{[b]}$
perform *backpropagation*
update $W^{(l)}$ and $b^{(l)}$

return trained NN



Neural Network Training

Mini-batch gradient descent

Define:

- **iteration**: one pass of mini-batch GD
- **epoch**: one pass over the dataset

Optimisation

Q: how to choose the size of the mini-batch?

Q: what happens if $\mathcal{B} = |\mathcal{D}|$?

animation by Luis Medina

Require: dataset $\mathcal{D} = \{(\vec{x}, y)\}$

Require: $\{\mathcal{D}_{[b]}\}_{b \in [1, \mathcal{B}]}$ s.t. $\bigcup_{i=0}^{\mathcal{B}-1} \mathcal{D}_{[b]} = \mathcal{D}$

for $0 \leq b < \mathcal{B}$ **do**

 compute forward pass on $\mathcal{D}_{[b]}$
 perform *backpropagation*
 update $W^{(l)}$ and $b^{(l)}$

return trained NN



Neural Network Training

Mini-batch gradient descent

Define:

- **iteration**: one pass of mini-batch GD
- **epoch**: one pass over the dataset

Optimisation

Q: how to choose the size of the mini-batch?

Even though it has a **regularisation** effect, I would not consider it as hyperparameter: it mostly depends on memory constraints.

Q: what happens if $\mathcal{B} = |\mathcal{D}|$?

This is called "stochastic" GD. Useful for huge datasets.

animation by Luis Medina

Require: dataset $\mathcal{D} = \{(\vec{x}, y)\}$

Require: $\{\mathcal{D}_{[b]}\}_{b \in [1, \mathcal{B}]}$ s.t. $\bigcup_{i=0}^{\mathcal{B}-1} \mathcal{D}_{[b]} = \mathcal{D}$

for $0 \leq b < \mathcal{B}$ **do**

 compute forward pass on $\mathcal{D}_{[b]}$
 perform *backpropagation*
 update $W^{(l)}$ and $b^{(l)}$

return trained NN



Neural Network Training

Advantages of mini-batch gradient descent

Honest question

Why should we use **mini-batch** gradient descent? What if my entire dataset fits into memory?

Smith et al. (2021)

Consider the gradient flow $\dot{\Theta} = f(\Theta)$ and the **discrete update**

$$\Theta_{t+1} = \Theta_t + \varepsilon f(\Theta_t) \quad \xleftrightarrow{\text{to be matched}} \quad \Theta(t + \varepsilon) \simeq \Theta(t) + \varepsilon f(\Theta(t))$$

Decompose $f(\Theta) = \sum_{n=0}^{\infty} \varepsilon^n f_{(n)}(\Theta)$. Then, we have, after n iterations with step size $\varepsilon = n\alpha$:

$$\begin{aligned}\Theta_{t+n} &= \Theta_t + \alpha f(\Theta_t) + \alpha f(\Theta_{t+1}) + \dots = \Theta_t + \alpha f(\Theta_t) + \alpha f(\Theta_t + \alpha f(\Theta_t)) + \dots \\ &= \Theta_t + n\alpha f_{(0)}(\Theta_t) + n^2\alpha^2 \left(f_{(1)}(\Theta_t) + \frac{n-1}{2n} \vec{\nabla} f_{(0)}(\Theta_t) \cdot f_{(0)}(\Theta_t) \right) + \dots\end{aligned}$$



Neural Network Training

Advantages of mini-batch gradient descent

Honest question

Why should we use **mini-batch** gradient descent? What if my entire dataset fits into memory?

Smith et al. (2021)

Consider the case of **gradient descent** (full):

$$f_{(0)}(\Theta) = -\vec{\nabla} \mathcal{L}(\Theta) \quad \text{s.t.} \quad \Theta_{t+1} = \Theta_t - \varepsilon \vec{\nabla} \mathcal{L}(\Theta)$$

Then $n \rightarrow \infty$ we need to introduce a counterterm if we proceed “step-by-step”:

does “renormalisation” ring a bell?

$$\Theta(t+\varepsilon) = \Theta(t) - \varepsilon \mathcal{L}(\Theta) \quad \Leftrightarrow \quad f_{(1)}(\Theta) = -\frac{1}{4} \vec{\nabla} \left\| \vec{\nabla} \mathcal{L}(\Theta) \right\|_2^2 \quad \Rightarrow \quad \mathcal{L}(\Theta) \leftarrow \mathcal{L}(\Theta) + \frac{1}{4} \left\| \vec{\nabla} \mathcal{L}(\Theta) \right\|_2^2$$



Neural Network Training

Advantages of mini-batch gradient descent

Honest question

Why should we use **mini-batch** gradient descent? What if my entire dataset fits into memory?

Smith et al. (2021)

Consider now the **mini-batch** loss:

$$f_{(0)} = \widehat{\mathcal{L}}(\Theta) = \frac{1}{B} \sum_{i=0}^{B-1} \mathcal{L}^{(i)}(\Theta) = \frac{1}{B} \sum_{i=0}^{B-1} \frac{1}{|B|} \sum_{k \in B} \mathcal{L}^k(\Theta)$$

and compute the **discrete update** over one epoch ($n = |B|$):

$$\begin{aligned}\Theta_B &= \Theta_0 - \varepsilon \vec{\nabla} \mathcal{L}^{(0)}(\Theta_0) - \varepsilon \vec{\nabla} \mathcal{L}^{(1)}(\Theta_1) - \varepsilon \vec{\nabla} \mathcal{L}^{(2)}(\Theta_2) + \dots \\ &= \Theta_0 - \varepsilon \sum_{i=0}^{B-1} \vec{\nabla} \mathcal{L}^{(i)}(\Theta_0) + \varepsilon^2 \sum_{i=0}^{B-1} \sum_{j < i} \vec{\nabla} \vec{\nabla} \mathcal{L}^{(i)}(\Theta_0) \cdot \vec{\nabla} \mathcal{L}^{(j)}(\Theta_0) + \dots\end{aligned}$$



Neural Network Training

Advantages of mini-batch gradient descent

Honest question

Why should we use **mini-batch** gradient descent? What if my entire dataset fits into memory?

Smith et al. (2021)

After one **epoch** is $\varepsilon \ll 1$, the mini-batch update does not introduce **noise**. However, if ε is finite, we have a $O(\varepsilon^2)$ term to keep in mind:

$$\mathbb{E} [\Theta_B] = \Theta_0 - \varepsilon B \vec{\nabla} \mathcal{L}(\Theta_0) + \frac{B^2 \varepsilon^2}{4} \vec{\nabla} \left(\left\| \vec{\nabla} \mathcal{L}(\Theta_0) \right\|_2^2 - \frac{1}{B^2} \sum_{i=1}^B \left\| \vec{\nabla} \mathcal{L}^{(i)}(\Theta_0) \right\|_2^2 \right) + O(B^3 \varepsilon^3)$$



Neural Network Training

Advantages of mini-batch gradient descent

Honest question

Why should we use **mini-batch** gradient descent? What if my entire dataset fits into memory?

Smith et al. (2021)

The first term in the parenthesis comes from the correction to the gradient descent $f_{(1)}$, but there is **one additional term!**

Let us recover the continuous update (the gradient flow):

$$\mathbb{E} [\Theta_B] \simeq \Theta(B\varepsilon) \quad \Leftrightarrow \quad \mathcal{L}(\Theta) \leftarrow \mathcal{L}(\Theta) + \frac{1}{4} \left\| \vec{\nabla} \mathcal{L}(\Theta) \right\|_2^2 + \frac{1}{4B} \sum_{i=0}^{B-1} \left\| \vec{\nabla} \mathcal{L}^{(i)}(\Theta) \right\|_2^2$$

The last is a **regularisation term** added “automatically” by **mini-batch gradient descent!**



Neural Network Training

Optimisation

The naive GD is good but can be improved:

- weight update might get stuck
- weight update might be too slow

A simple example

Let

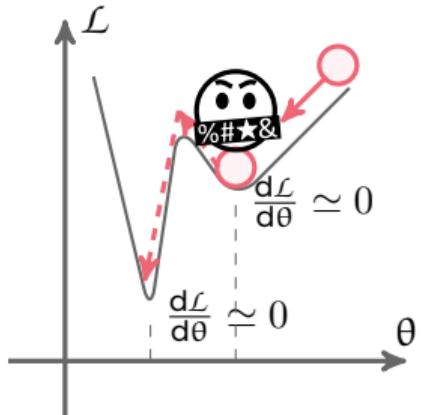
$$\mathcal{L}(\vec{\theta}) = \frac{1}{2} (\lambda_1 \theta_1^2 + \lambda_2 \theta_2^2), \quad 0 < \lambda_1 < \lambda_2,$$

s.t. $\vec{\nabla} \mathcal{L}(\vec{\theta}) = (\lambda_1 \theta_1, \lambda_2 \theta_2)$ to compute

$$\vec{\theta}^{(t+1)} = \vec{\theta}^{(t)} - \alpha \vec{\nabla} \mathcal{L}(\vec{\theta}), \quad \alpha > 0,$$

in order to find $\vec{\theta}^* = \vec{0}$.

see [Waldspurger \(CEREMADE\)](#)





Neural Network Training

Optimisation

The naive GD is good but can be improved:

- weight update might get stuck
- weight update might be too slow

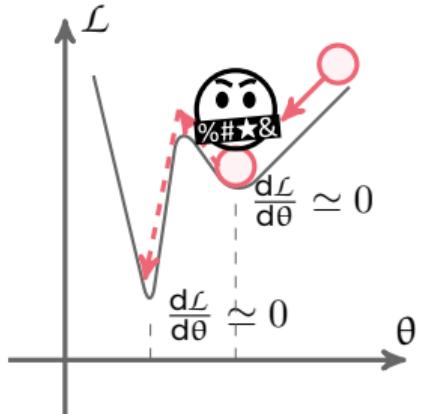
A simple example

We would like

$$\vec{\theta}^{(t+1)} = \left((1 - \alpha\lambda_1)\theta_1^{(t)}, (1 - \alpha\lambda_2)\theta_2^{(t)} \right)$$

s.t. $|1 - \alpha\lambda_i| \ll 1$, for $i = 1, 2$.

see [Waldspurger \(CEREMADE\)](#)





Neural Network Training

Optimisation

The naive GD is good but can be improved:

- weight update might get stuck
- weight update might be too slow

A simple example

We would like

$$\vec{\theta}^{(t+1)} = \left((1 - \alpha\lambda_1)\theta_1^{(t)}, (1 - \alpha\lambda_2)\theta_2^{(t)} \right)$$

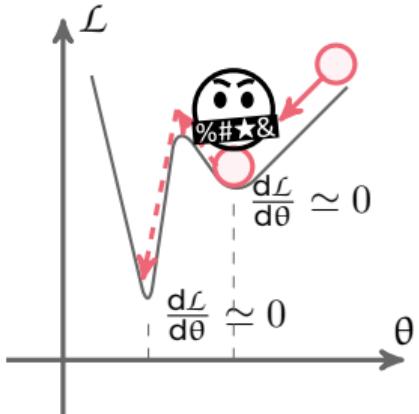
s.t. $|1 - \alpha\lambda_i| \ll 1$, for $i = 1, 2$.

However,

$$\alpha = O(\lambda_1^{-1}) \Rightarrow 1 - \alpha\lambda_2 = 1 - \frac{\lambda_2}{\lambda_1} < 0$$

and the update of θ_2 diverges.

see Waldspurger (CEREMADE)





Neural Network Training

Optimisation

The naive GD is good but can be improved:

- weight update might get stuck
- weight update might be too slow

A simple example

We would like

$$\vec{\theta}^{(t+1)} = \left((1 - \alpha\lambda_1)\theta_1^{(t)}, (1 - \alpha\lambda_2)\theta_2^{(t)} \right)$$

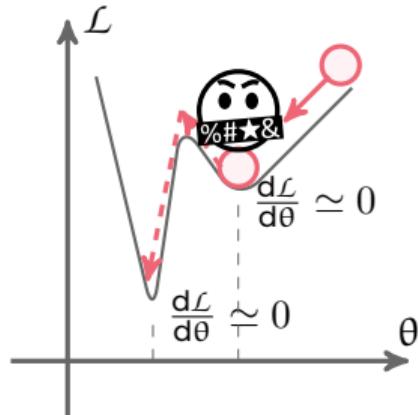
s.t. $|1 - \alpha\lambda_i| \ll 1$, for $i = 1, 2$.

And

$$\alpha = O(\lambda_2^{-1}) \Rightarrow 1 - \alpha\lambda_1 = 1 - \frac{\lambda_1}{\lambda_2} \ll 1$$

and the update of θ_1 is slow.

see [WalDSPURGER \(CEREMADE\)](#)



REMARKS

1. is the loss *landscape* still “nice”?
2. are all “nice” losses still **convex**?





Neural Network Training

Gradient descent and momentum

Introduce the GD algorithm with **momentum** (Ω set of weights and biases):



Neural Network Training

Gradient descent and momentum

Introduce the GD algorithm with **momentum** (Ω set of weights and biases):

The “Heavy Ball” algorithm | Polyak’s Momentum (see Polyak (1964))

Require: $\alpha \in \mathbb{R}^+, \Omega^{(0)}, \mathcal{L}, T \in \mathbb{N} \setminus \{0\}, \vec{m}^{(0)} = \vec{0}$

for $0 \leq t < T$ **do**

$$\vec{G}^{(t)} \leftarrow \vec{\nabla} \mathcal{L}(\Omega^{(t)})$$

$$\vec{m}^{(t+1)} \leftarrow \gamma \vec{m}^{(t)} + (1 - \gamma) \vec{G}^{(t)}$$

▷ momentum

▷ “educated” steepest descent

return $\Omega^{(T)}$



Neural Network Training

Gradient descent and momentum

Introduce the GD algorithm with **momentum** (Ω set of weights and biases):

The “Heavy Ball” algorithm | Polyak’s Momentum (see Polyak (1964))

Require: $\alpha \in \mathbb{R}^+, \Omega^{(0)}, \mathcal{L}, T \in \mathbb{N} \setminus \{0\}, \vec{m}^{(0)} = \vec{0}$

for $0 \leq t < T$ **do**

$$\vec{G}^{(t)} \leftarrow \vec{\nabla} \mathcal{L}(\Omega^{(t)})$$

$$\vec{m}^{(t+1)} \leftarrow \gamma \vec{m}^{(t)} + (1 - \gamma) \vec{G}^{(t)}$$

$$\Omega^{(t+1)} \leftarrow \Omega^{(t)} - \alpha \vec{m}^{(t+1)}$$

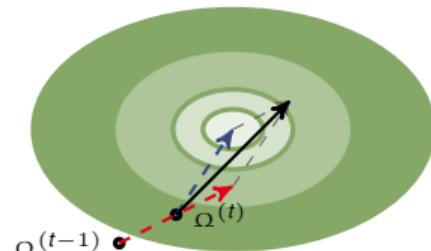
return $\Omega^{(T)}$

▷ momentum
▷ “educated” steepest descent

This can be equivalently expressed by:

$$\Omega^{(t+1)} = \Omega^{(t)} - \tilde{\alpha} \vec{\nabla} f(\Omega^{(t)}) + \tilde{\beta} (\Omega^{(t)} - \Omega^{(t-1)}),$$

where $\tilde{\alpha} = \alpha(1 - \gamma)$ and $\tilde{\beta} = \frac{\alpha\gamma}{\alpha-1}$.





Neural Network Training

Gradient descent and momentum

Introduce the GD algorithm with **momentum** (Ω set of weights and biases):

ADA(ptive) M(omentum estimation) (see Kingma and Ba (2014))

Require: $\alpha \in \mathbb{R}^+, \theta^{(0)}, \mathcal{L}, T \in \mathbb{N} \setminus \{0\}, \vec{m}^{(0)} = \vec{0}, \vec{v}^{(0)} = \vec{0}$

for $0 \leq t < T$ **do**

$$\vec{G}^{(t)} \leftarrow \nabla \mathcal{L}(\Omega^{(t)})$$

$$\vec{m}^{(t+1)} \leftarrow \beta_1 \vec{m}^{(t)} + (1 - \beta_1) \vec{G}^{(t)} \quad \triangleright \text{first momentum estimate } (\beta_1 = 0.9)$$

$$\vec{v}^{(t+1)} \leftarrow \beta_2 \vec{v}^{(t)} + (1 - \beta_2) \left(\vec{G}^{(t)} \right)^2 \quad \triangleright \text{second momentum estimate } (\beta_2 = 0.999)$$

$$\hat{\vec{m}}^{(t+1)} = \frac{\vec{m}^{(t+1)}}{1 - \beta_1^t}$$

$$\hat{\vec{v}}^{(t+1)} = \frac{\vec{v}^{(t+1)}}{1 - \beta_2^t}$$

$$\Omega^{(t+1)} \leftarrow \Omega^{(t)} - \alpha \frac{\hat{\vec{m}}^{(t+1)}}{\sqrt{\hat{\vec{v}}^{(t+1)}} + \epsilon} \quad \triangleright \text{momenta-aware steepest descent}$$

return $\Omega^{(T)}$



Neural Network Regularisation

Weight decay

Idea: avoid large parameter updates when in advanced training

Response: add an (exponential) “weight decay” term in the optimisation, proportional to the magnitude of the parameters themselves:

$$\Omega^{(t+1)} = (1 - \eta) \Omega^{(t)} - \alpha \frac{\partial \mathcal{L}(\Omega^{(t)})}{\partial \Omega^{(t)}}, \quad \eta > 0.$$

(see also Loshchilov and Hutter (2017))



Neural Network Regularisation

Weight decay

Idea: avoid large parameter updates when in advanced training

Response: add an (exponential) “weight decay” term in the optimisation, proportional to the magnitude of the parameters themselves:

$$\Omega^{(t+1)} = (1 - \eta) \Omega^{(t)} - \alpha \frac{\partial \mathcal{L}(\Omega^{(t)})}{\partial \Omega^{(t)}}, \quad \eta > 0.$$

The *regularisation* helps reducing the magnitude of the parameters during training.

THINK

- Is weight decay equivalent to a L_2 regularisation term in **vanilla** GD?

$$\mathcal{L}(\Omega) \leftarrow \mathcal{L}(\Omega) + \frac{\eta}{2} \|\Omega\|_2^2$$

(see also Loshchilov and Hutter (2017))



Neural Network Regularisation

Weight decay

Idea: avoid large parameter updates when in advanced training

Response: add an (exponential) “weight decay” term in the optimisation, proportional to the magnitude of the parameters themselves:

$$\Omega^{(t+1)} = (1 - \eta) \Omega^{(t)} - \alpha \frac{\partial \mathcal{L}(\Omega^{(t)})}{\partial \Omega^{(t)}}, \quad \eta > 0.$$

The *regularisation* helps reducing the magnitude of the parameters during training.

THINK

- Is weight decay equivalent to a L_2 regularisation term in **vanilla** GD?

$$\mathcal{L}(\Omega) \leftarrow \mathcal{L}(\Omega) + \frac{\eta}{2} \|\Omega\|_2^2$$

Technically, iff. $\eta \leftarrow \eta \alpha^{-1}$, but this is usually ignored, so, yes!

(see also Loshchilov and Hutter (2017))



Neural Network Regularisation

Weight decay

Idea: avoid large parameter updates when in advanced training

Response: add an (exponential) “weight decay” term in the optimisation, proportional to the magnitude of the parameters themselves:

$$\Omega^{(t+1)} = (1 - \eta) \Omega^{(t)} - \alpha \frac{\partial \mathcal{L}(\Omega^{(t)})}{\partial \Omega^{(t)}}, \quad \eta > 0.$$

The *regularisation* helps reducing the magnitude of the parameters during training.

THINK

- Think about momentum-GD (e.g. ADAM or SGD). Is L_2 regularisation still equivalent to weight decay? Remember that the regularisation leads to

$$\vec{g}^{(t)} = \vec{\nabla} \mathcal{L}(\Omega^{(t)}) + \eta \Omega^{(t)}$$

in the algorithm. Can you think of a straightforward modification to recover the “correct” weight decay?



4 Neural Networks

- Regularisation of neural networks

Table of contents

1. Some History and Philosophy to Start

2. The ML Mindset

3. ML Algorithms

4. Neural Networks

Computational graphs

Non Linearity of Neural Networks

Approximation theorems

Neural network training

Regularisation of neural networks

5. Conclusions





Neural Network Regularisation

Batch normalisation

Is there a way to contain the growth/variation of the layers?

Idea: eliminate/reduce the *internal covariate shift*, i.e. the **change of distribution** of the activated outputs $y^{(\ell)} = a^{(\ell)} \odot z^{(\ell)}$

see Ioffe and Szegedy (2015)



Neural Network Regularisation

Batch normalisation

Is there a way to contain the growth/variation of the layers?

Idea: eliminate/reduce the *internal covariate shift*, i.e. the **change of distribution** of the activated outputs $y^{(\ell)} = a^{(\ell)} \odot z^{(\ell)}$

Batch normalisation (training phase)

Introduce $\forall \ell = 1, 2, \dots, N$ and $\forall b = 1, 2, \dots, \mathcal{B}$:

$$\mu_{[b]}^{(\ell)} = \mathbb{E}_{\mathcal{D}_{[b]}} [z^{(\ell)}] = \frac{1}{|\mathcal{D}_{[b]}|} \sum_{i=1}^{|\mathcal{D}_{[b]}|} z_i^{(\ell)}, \quad \left(\sigma_{[b]}^{(\ell)} \right)^2 = \text{Var}_{\mathcal{D}_{[b]}} (z^{(\ell)}) = \frac{1}{|\mathcal{D}_{[b]}|} \sum_{i=1}^{|\mathcal{D}_{[b]}|} \left(z_i^{(\ell)} - \mu_{[b]}^{(\ell)} \right)^2$$

see Ioffe and Szegedy (2015)



Neural Network Regularisation

Batch normalisation

Is there a way to contain the growth/variation of the layers?

Idea: eliminate/reduce the *internal covariate shift*, i.e. the **change of distribution** of the activated outputs $y^{(\ell)} = a^{(\ell)} \odot z^{(\ell)}$

Batch normalisation (training phase)

Normalize $\forall \ell = 1, 2, \dots, N$ and $\forall b = 1, 2, \dots, \mathcal{B}$:

$$\hat{z}_{[b]}^{(\ell)} = \frac{z^{(\ell)} - \mu_{[b]}^{(\ell)}}{\sigma_{[b]}^{(\ell)} + \varepsilon}$$

see Ioffe and Szegedy (2015)



Neural Network Regularisation

Batch normalisation

Is there a way to contain the growth/variation of the layers?

Idea: eliminate/reduce the *internal covariate shift*, i.e. the **change of distribution** of the activated outputs $y^{(\ell)} = a^{(\ell)} \odot z^{(\ell)}$

Batch normalisation (training phase)

Interpolate $\forall \ell = 1, 2, \dots, N$ and $\forall b = 1, 2, \dots, \mathcal{B}$:

$$\text{BN}_{[b]}^{(\ell)} \left(\hat{z}^{(\ell)}; \gamma, \beta \right) = \gamma_{[b]} \hat{z}_{[b]}^{(\ell)} + \beta_{[b]},$$

where $\gamma^{(\ell)}$ and $\beta^{(\ell)}$ are **learnable** scalars

see Ioffe and Szegedy (2015)



Neural Network Regularisation

Batch normalisation

Is there a way to contain the growth/variation of the layers?

Idea: eliminate/reduce the *internal covariate shift*, i.e. the **change of distribution** of the activated outputs $y^{(\ell)} = a^{(\ell)} \odot z^{(\ell)}$

Batch normalisation (training phase)

Replace $\forall \ell = 1, 2, \dots, N$ and $\forall b = 1, 2, \dots, \mathcal{B}$:

$$y^{(\ell)} = a^{(\ell)} \odot z^{(\ell)} \quad \rightarrow \quad \hat{y}_{[b]}^{(\ell)} = a^{(\ell)} \odot \text{BN}_{[b]}^{(\ell)} \left(\hat{z}^{(\ell)}; \gamma, \beta \right)$$

see Ioffe and Szegedy (2015)



Neural Network Regularisation

Batch normalisation

Is there a way to contain the growth/variation of the layers?

Idea: eliminate/reduce the *internal covariate shift*, i.e. the **change of distribution** of the activated outputs $y^{(\ell)} = a^{(\ell)} \odot z^{(\ell)}$

Batch normalisation (inference phase)

Inference **must not** depend on batch size!

Compute (after training) $\forall \ell = 1, 2, \dots, N$:

$$\mu^{(\ell)} = \frac{1}{\mathcal{B}} \sum_{b=1}^{\mathcal{B}} \mu_{[b]}^{(\ell)}, \quad (\sigma^{(\ell)})^2 = \frac{1}{\mathcal{B}} \sum_{b=1}^{\mathcal{B}} (\mu_{[b]}^{(\ell)} - \mu^{(\ell)})^2.$$

N.B.: the original paper uses the *unbiased estimate* of the variance.

see Ioffe and Szegedy (2015)



Neural Network Regularisation

Batch normalisation

Is there a way to contain the growth/variation of the layers?

Idea: eliminate/reduce the *internal covariate shift*, i.e. the **change of distribution** of the activated outputs $y^{(\ell)} = a^{(\ell)} \odot z^{(\ell)}$

Batch normalisation (inference phase)

Inference **must not** depend on batch size!

Replace all $\text{BN}^{(\ell)}$ operations $\forall \ell = 1, 2, \dots, N$:

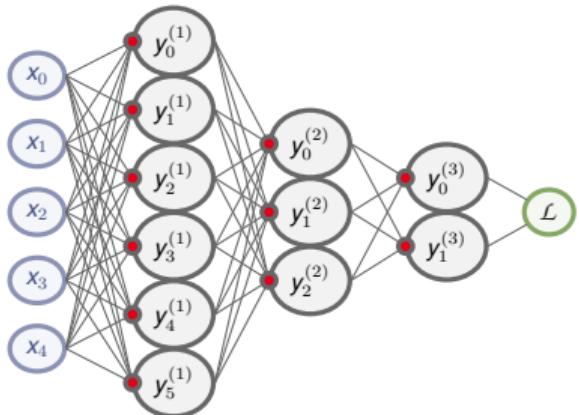
$$\text{BN}_{[b]}^{(\ell)}(\cdot; \gamma, \beta) \rightarrow \frac{\gamma}{\sigma^{(\ell)} + \varepsilon} z^{(\ell)} + \left(\beta - \frac{\gamma \mu^{(\ell)}}{\sigma^{(\ell)} + \varepsilon} \right)$$

see Ioffe and Szegedy (2015)



Neural Network Regularisation

Dropout



Some remarks:

- NNs are **high variance** models
- some paths might be strongly correlated (**co-adaptation**)

For the sake of simplicity, consider the following regression **linear model** (homogeneous) for $i = 1, 2, \dots, h$:

$$y_i = f_i(\vec{x}) = \sum_{j=1}^k W_{ij} x_j$$

and let $Q \in \{0, 1\}^{h \times k}$ a matrix of Bernoulli variables

$$\mathbb{P}(Q_{ij} = 1) = p = 1 - \mathbb{P}(Q_{ij} = 0),$$

for $i = 1, 2, \dots, h$, and $j = 1, 2, \dots, k$.
In other words, $\mathbb{E}[Q_{ij}] = p$, and
 $\text{Var}(Q_{ij}) = p(1 - p)$:

(see Wager et al. (2013))



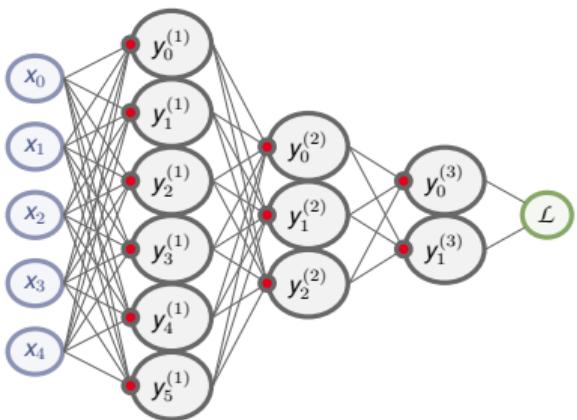
Neural Network Regularisation

Dropout

Then, define the “dropout model”:

$$\tilde{y}_i = \tilde{f}_i(\vec{x}) = ((Q \odot W) \vec{x})_i = \sum_{j=1}^k Q_{ij} W_{ij} x_j$$

This is equivalent to any hidden layer in a NN:



Some remarks:

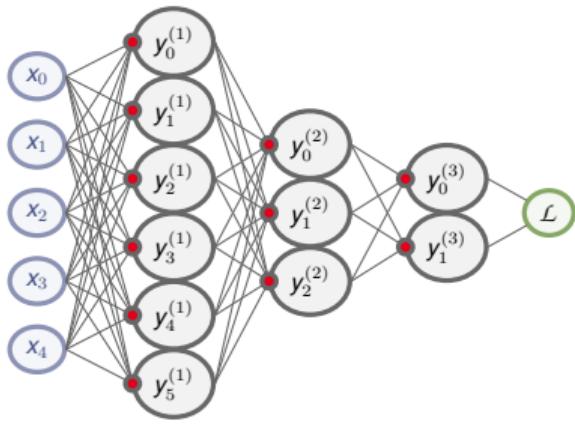
- NNs are **high variance** models
- some paths might be strongly correlated (**co-adaptation**)





Neural Network Regularisation

Dropout



We can compute the loss

$$\mathcal{L}(y, \tilde{y}) = \frac{1}{2n} \sum_{i=1}^n \sum_{j=1}^k (y_i - Q_{ij} W_{ij} x_j)^2,$$

whose gradients are:

$$\begin{aligned}\frac{\partial \tilde{\mathcal{L}}}{\partial W_{ij}} &= -Q_{ij} x_j \left(y_i - \sum_{t=1}^k Q_{it} W_{it} x_t \right) \\ &= -Q_{ij} y_i x_j + Q_{ij}^2 W_{ij} x_j^2 + \sum_{t=1, t \neq j}^k Q_{ij} Q_{it} W_{it} x_j x_t\end{aligned}$$

(see Wager et al. (2013))

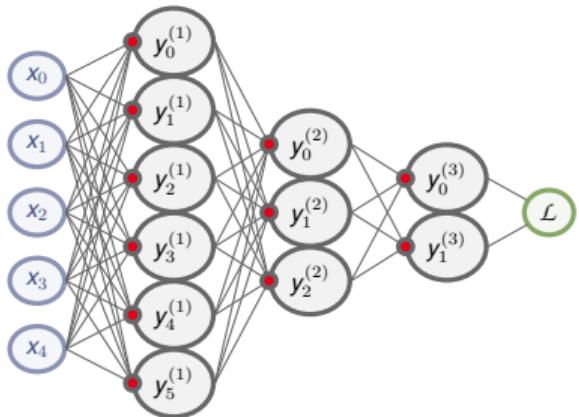
Some remarks:

- NNs are **high variance** models
- some paths might be strongly correlated (**co-adaptation**)



Neural Network Regularisation

Dropout



Some remarks:

- NNs are **high variance** models
- some paths might be strongly correlated (**co-adaptation**)

Let us compute the average value of the gradients over the **dropout** distribution:

$$\begin{aligned}\mathbb{E} \left[\frac{\partial \tilde{\mathcal{L}}}{\partial W_{ij}} \right] &= -\mathbb{E} [Q_{ij}] y_i x_j + \mathbb{E} [Q_{ij}^2] W_{ij} x_j^2 \\ &\quad + \sum_{t=1, t \neq j}^k \mathbb{E} [Q_{ij}] \mathbb{E} [Q_{it}] W_{it} x_j x_t.\end{aligned}$$

Remember that $\mathbb{E} [X^2] = \text{Var}(X) + \mathbb{E} [X]^2$:

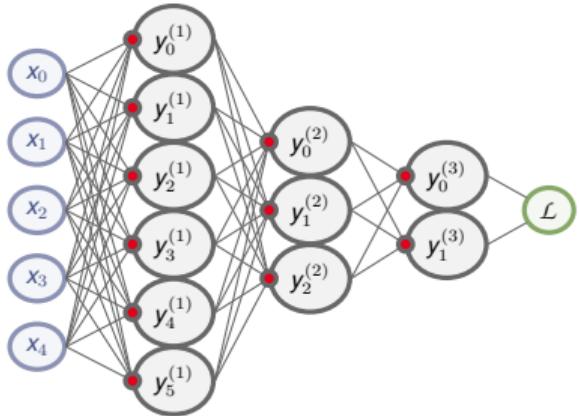
$$\begin{aligned}\mathbb{E} \left[\frac{\partial \tilde{\mathcal{L}}}{\partial W_{ij}} \right] &= -py_i x_j + p^2 W_{ij} x_j^2 + p(1-p) W_{ij} x_j^2 \\ &\quad + p^2 \sum_{t=1, t \neq j}^k W_{it} x_j x_t.\end{aligned}$$

(see Wager et al. (2013))



Neural Network Regularisation

Dropout



Some remarks:

- NNs are **high variance** models
- some paths might be strongly correlated (**co-adaptation**)

Though perturbed by constants p and p^2 , we can reconstruct the usual loss + a regularisation:

$$\mathbb{E} \left[\frac{\partial \tilde{\mathcal{L}}}{\partial W_{ij}} \right] \sim \mathbb{E} \left[\frac{\partial \mathcal{L}_p}{\partial W_{ij}} \right] + p(1-p) W_{ij} x_j^2.$$

Dropout

Dropout is a “feature noising” **regularisation** technique, which **can** be applied to **NNs** to prevent **overfitting** and **co-adaption**.

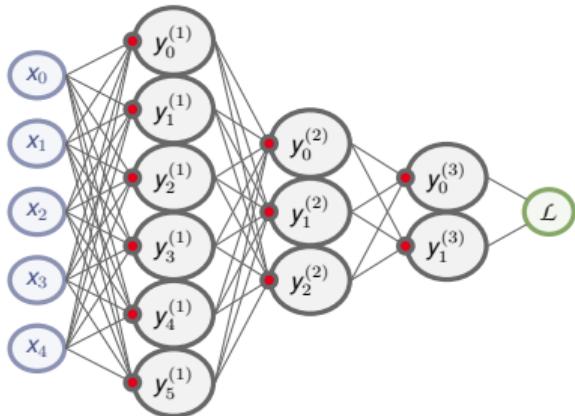
Dropout replaced by Id in inference (usually).

(see Wager et al. (2013))



Neural Network Regularisation

Dropout



Dropout

Dropout is a “feature noise” **regularisation** technique, which **can** be applied to **NNs** to prevent **overfitting** and **co-adaption**.

The elegant idea is to average the outputs of the models over a **noise** component ξ . Consider an exponential family of likelihood functions, and take the log-partition function A :

$$\mathcal{L} \left(\mathbb{E}_{\xi} [A(\Omega_{\xi}, x)] \right) = \mathcal{L}(A(\Omega, x)) + R(\Omega),$$

where the regularisation

$$R(\Omega) \sim \mathbb{E}_{\xi} [A(\Omega_{\xi}, x)] - A(\Omega, x) \simeq \frac{1}{2} A''(\Omega, x) \text{Var}_{\xi} (A(\Omega_{\xi}, x)),$$

in the case $\mathbb{E}_{\xi} [\Omega_{\xi}] = \Omega$.

(see Wager et al. (2013))



5. Conclusions

Concluding remarks

Table of contents

1. Some History and Philosophy to Start
2. The ML Mindset
3. ML Algorithms
4. Neural Networks
5. **Conclusions**
Concluding remarks



Conclusions

A quick summary

In summary, you should now have a good understanding of:

- what **ML** is, and what are its **principles**
- how to work with **ML pipelines** for high quality research
- how to perform **validation** of ML models
- how to **evaluate** the performance of ML models
- what is the “**variance vs bias**” trade-off
- what a **loss function** is, and what its purpose
- several **regularisation** techniques
- different **unsupervised and supervised techniques** (including ensembles)
- what a **NN** is and how to train one

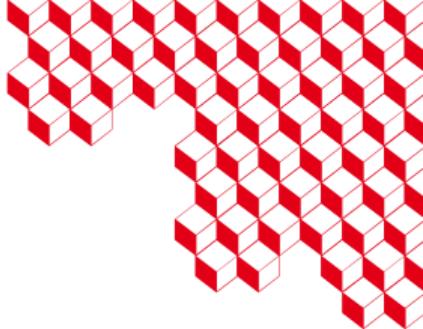
Hoping that I did not bore anyone to death...



Bibliography I

Some personal favourites

- [1] F. Chollet, *Deep Learning with Python*. Manning Publications Co., 2018, 384 pp., isbn: 978-1-61729-443-3. [Online]. Available: <https://www.manning.com/books/deep-learning-with-python>.
- [2] A. J. Ferreira and M. A. T. Figueiredo, "Boosting Algorithms: A Review of Methods, Theory, and Applications," in *Ensemble Machine Learning: Methods and Applications*, C. Zhang and Y. Ma, Eds., New York, NY: Springer, 2012, pp. 35–85, isbn: 978-1-4419-9326-7. doi: 10.1007/978-1-4419-9326-7_2.
- [3] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. 2019, isbn: 978-1-4920-3264-9. [Online]. Available: <http://shop.oreilly.com/product/0636920142874.do>.
- [4] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: <https://www.deeplearningbook.org/>.
- [5] C. Gruber, P. O. Schenk, M. Schierholz, et al., "Sources of Uncertainty in Machine Learning – A Statisticians' View," *arXiv: 2305.16703 [cs, stat]*. (2023), pre-published.
- [6] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning* (Springer Series in Statistics). New York, NY: Springer New York, 2009, isbn: 978-0-387-84857-0 978-0-387-84858-7. doi: 10.1007/978-0-387-84858-7.
- [7] MIT Open Learning Library, "Introduction to Machine Learning," (2020), [Online]. Available: <https://openlearninglibrary.mit.edu/courses/course-v1:MITx+6.036+1T2019/about>.
- [8] P. Mehta, M. Lukic, C.-H. Wang, et al., "A high-bias, low-variance introduction to Machine Learning for physicists," *Physics Reports*, vol. 810, pp. 1–124, 2019, issn: 03701573. doi: 10.1016/j.physrep.2019.03.001. arXiv: 1803.08823 [physics.comp-ph].
- [9] S. S. Skiena, *The Data Science Design Manual*. Springer, 2017, isbn: 978-3-319-55444-0. [Online]. Available: <https://link.springer.com/book/10.1007/978-3-319-55444-0>.
- [10] Standford Online, "CS229: Machine Learning," (2024), [Online]. Available: <https://cs229.stanford.edu/>.
- [11] Standford Online, "Machine Learning Specialization," (2022), [Online]. Available: <https://online.stanford.edu/courses/soe-ymls-machine-learning-specialization>.
- [12] A. Zheng and A. Casari, *Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists*. O'Reilly Media, Inc., 2018, isbn: 978-1-4919-5324-2. [Online]. Available: <https://www.oreilly.com/library/view/feature-engineering-for/9781491953235>.



In bocca al lupo !

CEA SACLAY
91191 Gif-sur-Yvette Cedex
France
riccardo.finotello@cea.fr