

Artificial intelligence and chemistry

Tony Bonnaire

Image generated with Midjourney «A representation of deep learning merging with chemistry»

Organisation of the course

Introduction to ML

Linear models

SL principles

Trees and neural networks

Risk optimisation



Given by: Tony Bonnaire (+ Pablo Mas for the project)



Format: Lectures + hands-on sessions then data challenge (in chemistry)



Exam: Paper analysis (50%) + oral presentation for the challenge (50%)



Aim: Introduce you to the basics of ML principles and carry out a project



Some references:

- [Deep Learning: Foundations and Concepts](#), Bishop & Bishop, 2023,
- [Deep Learning](#), Goodfellow et al., 2016,
- [Deep Learning with Python](#), Chollet, 2016,
- [Learning Theory from First Principles](#), Francis Bach, 2024,
- <https://challengedata.ens.fr>: a bank of data science challenges to apply all the things we will learn in this course

Intelligent systems

Introduction to ML

Linear models

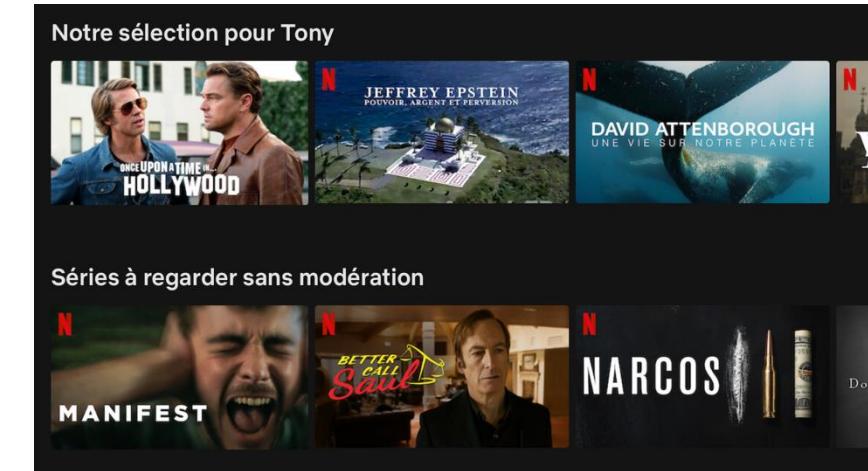
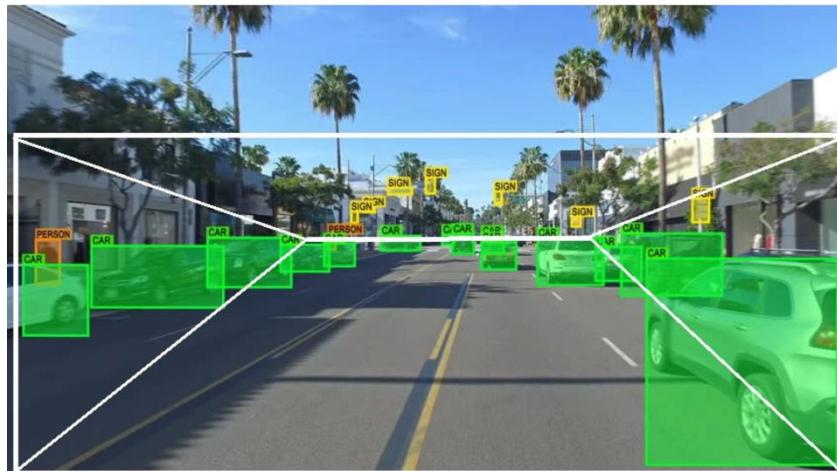
SL principles

Trees and neural networks

Risk optimisation

AI goal

Design **systems** capable of performing complex tasks requiring ***intelligence*** (i.e. using reasoning, perception or language) **to take decisions** and **make predictions**.



AI revolution

Introduction to ML

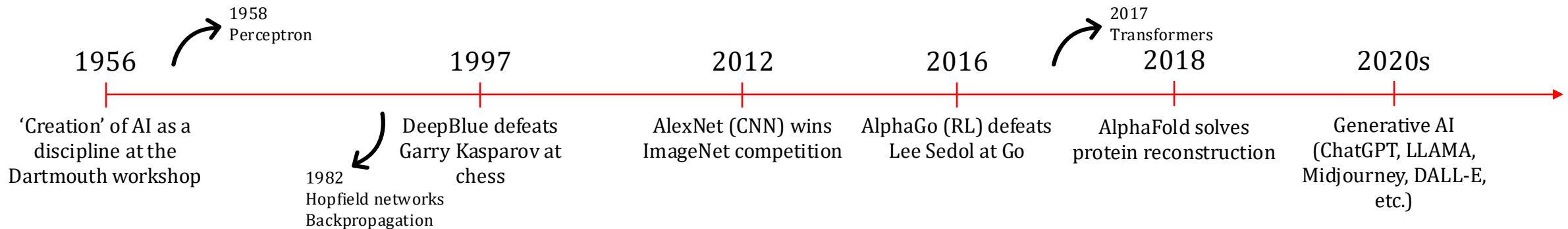
Linear models

SL principles

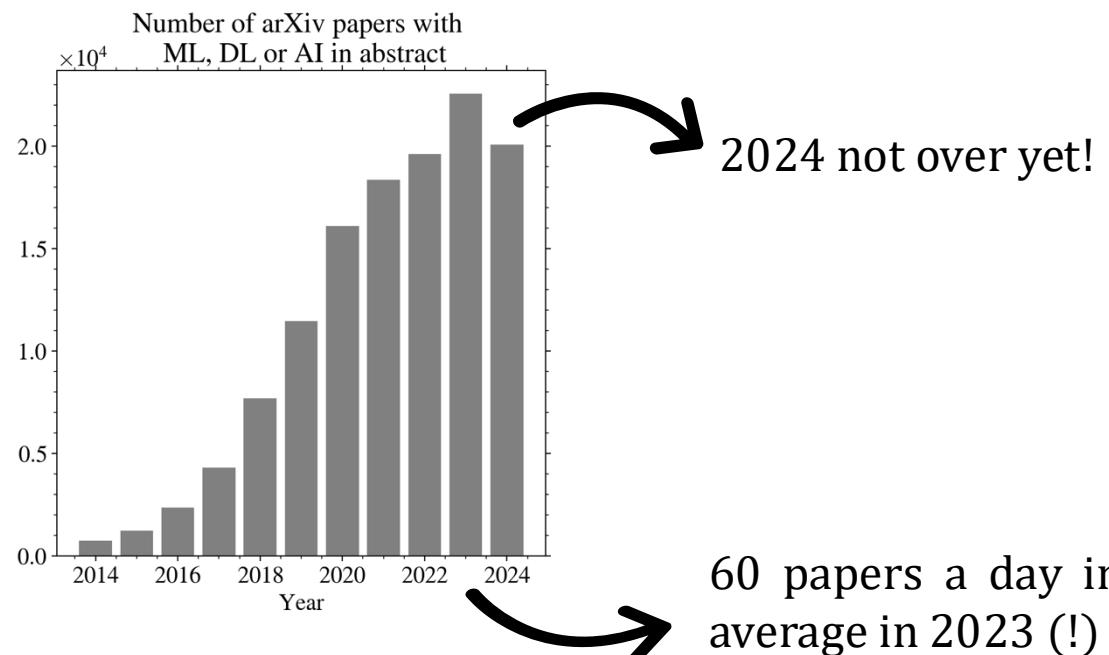
Trees and neural networks

Risk optimisation

Some (selected) AI breakthroughs



AI in science



Some scientific applications

Healthcare

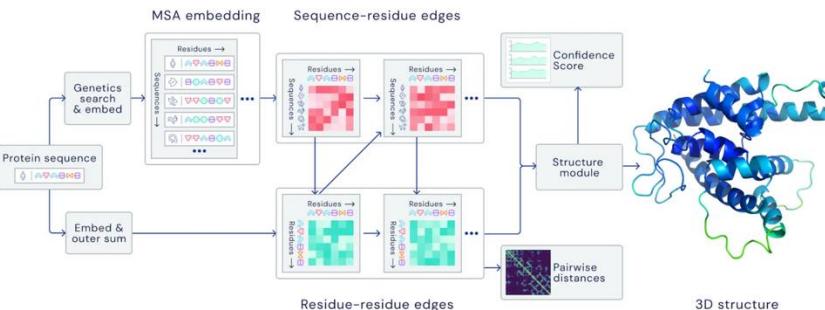
- Drug discovery
- Protein structure reconstruction

Astrophysics and cosmology

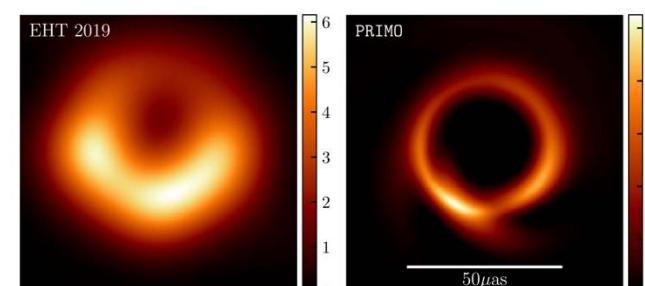
- Galaxy deblending
- Image restoration
- Source separation

Theoretical physics

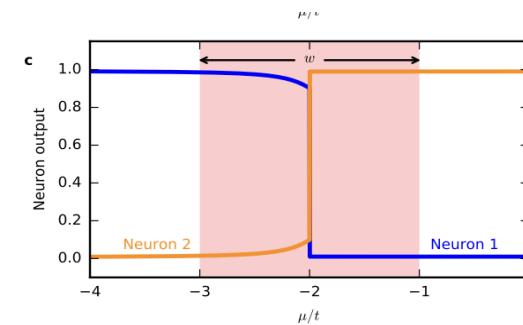
- Study phase transitions
- Discover experiments and equations



[Jumper et al., 2021](#)



[Medeiros et al., 2023](#)



[Van Nieuwenburg et al., 2017](#)

... And many more (climate forecast, fraud detection in cybersecurity, binding energies in quantum chemistry)

What is “learning”?

Introduction to ML

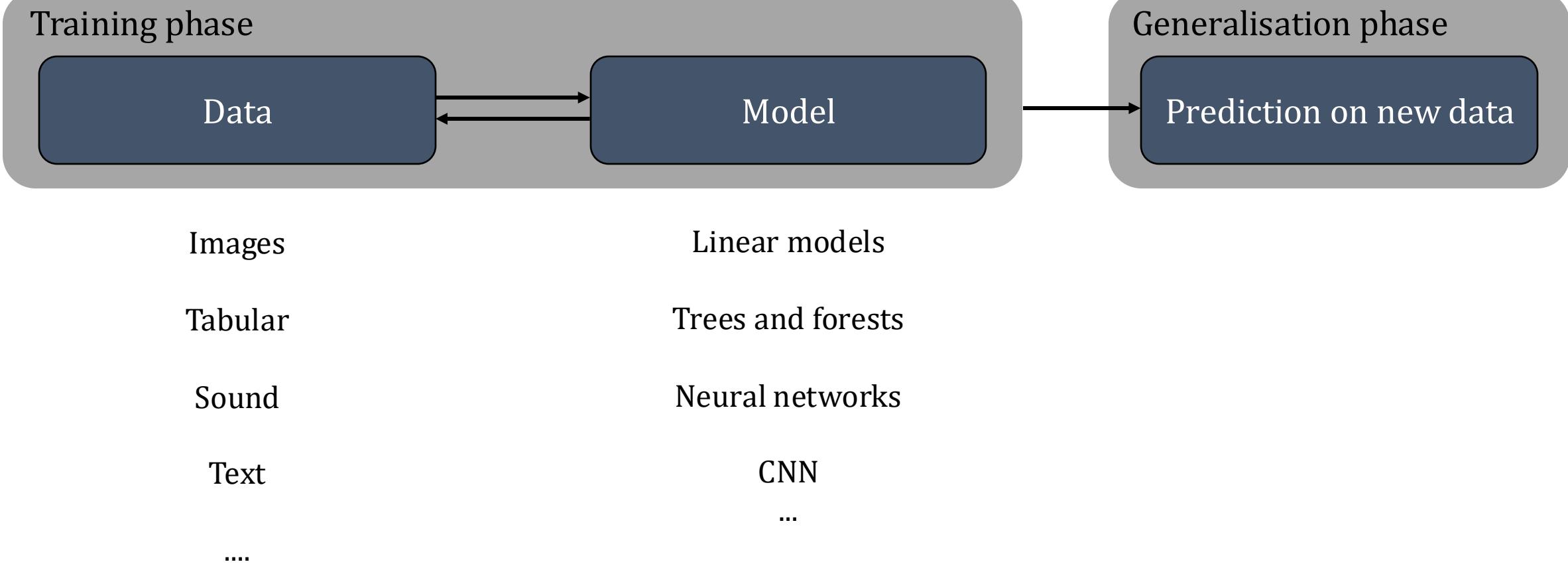
Linear models

SL principles

Trees and neural networks

Risk optimisation

Machine Learning came as a solution to design intelligent systems, replacing handcrafted decision rules by **learnt rules** using **training data** and **optimisation** of **parameterised models**.



What is “learning”? An example

Introduction to ML

Linear models

SL principles

Trees and neural networks

Risk optimisation

Training phase

Data

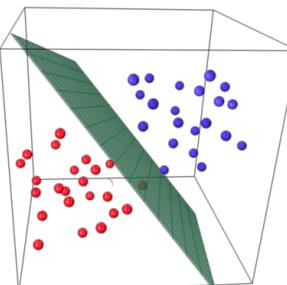
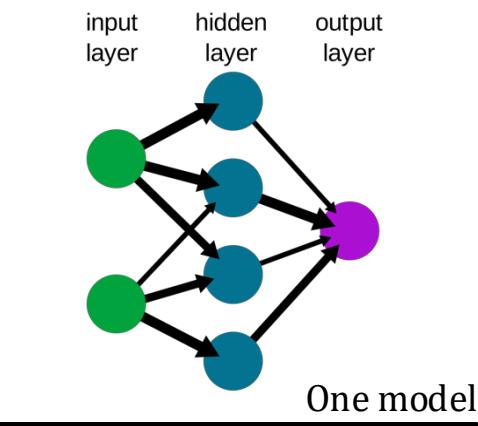
Model

Generalisation phase

Prediction on new data



Images of a “cat” or “dog”



Another model



“cat” or “dog”?

ML and the scientific method

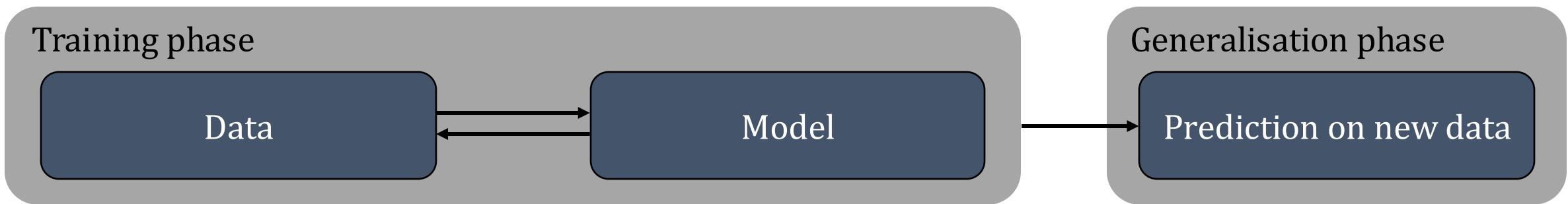
Introduction to ML

Linear models

SL principles

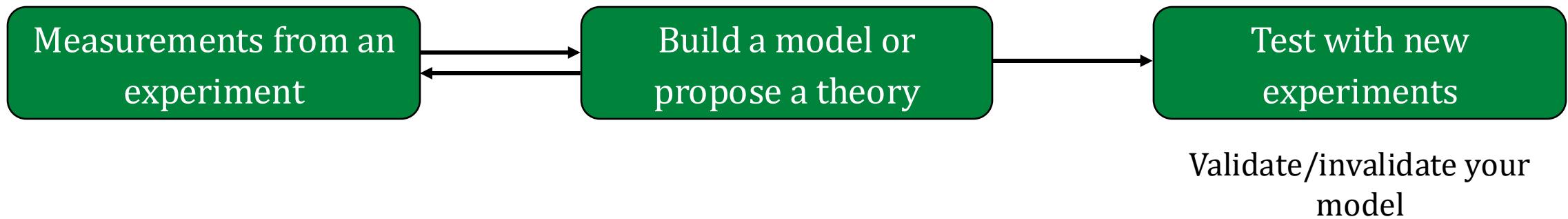
Trees and neural networks

Risk optimisation



...In fact, all this is close to what you know!

The scientific method



ML building blocks

Introduction to ML

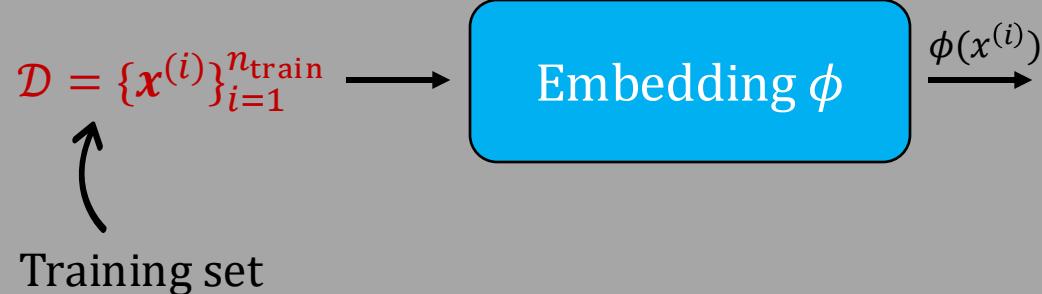
Linear models

SL principles

Trees and neural networks

Risk optimisation

Training phase



1. Data are **unstructured**, sometimes **noisy** and **unprocessed** like pixels of an image or sequence of characters or words.
2. The embedding $\phi(x^{(i)})$ is a **structured, numerical** vector representation of the data whose elements are **meaningful features** that depends on the data and the purpose. It can be **handcrafted or learnt**.

Finding a good embedding is a central part of ML: it eases the problem by preserving the essential structure of the data that matters for the task but makes it solvable using simple models.

ML building blocks

Introduction to ML

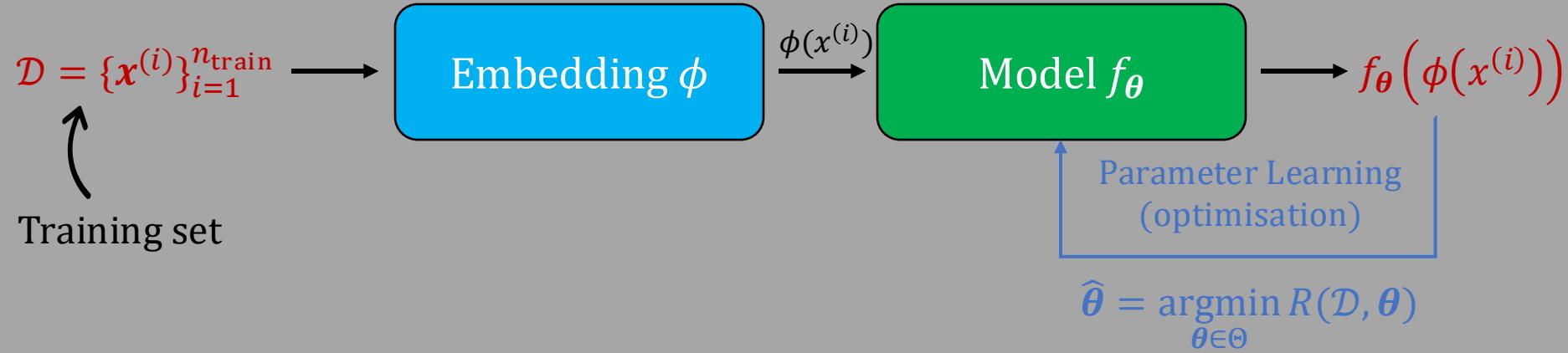
Linear models

SL principles

Trees and neural networks

Risk optimisation

Training phase

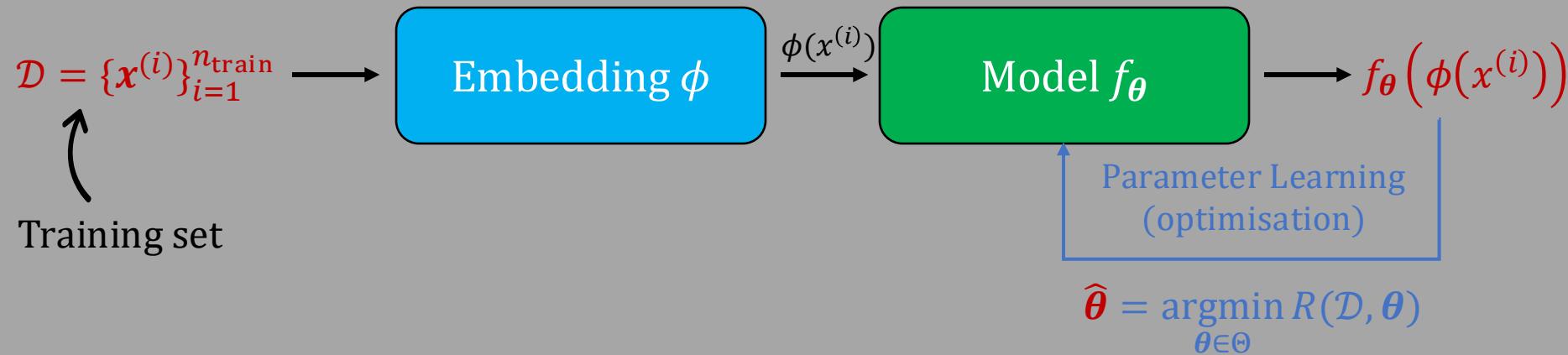


Some notations and terminologies:

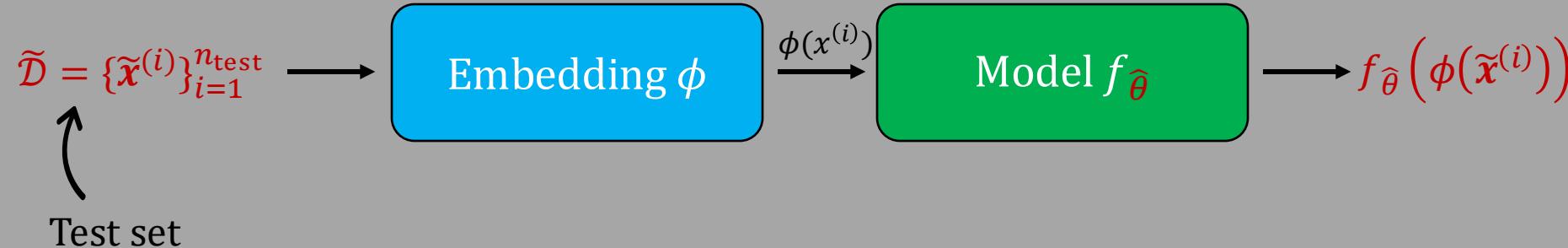
- $x^{(i)} \in \mathbb{R}^d$ is *one training data* (there are n_{train} of them),
- $\phi(x^{(i)}) \in \mathbb{R}^{d'}$ is an embedding of $x^{(i)}$ sometimes called ***feature vector***,
- $\theta \in \Theta \subset \mathbb{R}^p$ are the ***parameters*** of the model,
- $R(\mathcal{D}, \theta)$ is the ***risk*** and measures the error of the model with parameters θ on data X .

At the end of the training procedure, we have a model $f_{\hat{\theta}}$ committing an error of $R_{\text{train}} = R(\mathcal{D}, \hat{\theta})$ on the training set.

Training phase



Generalisation phase



 Using the test set, we can evaluate the test error $R_{\text{test}} = R(\tilde{\mathcal{D}}, \hat{\theta})$ and compare it to R_{train} to detect **generalisation issues (overfitting or underfitting)**.

Families of ML problems

Introduction to ML

Linear models

SL principles

Trees and neural networks

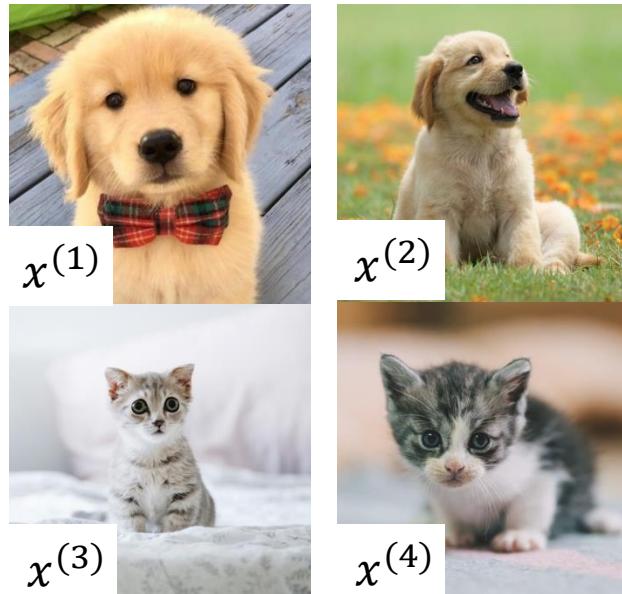
Risk optimisation

Supervised learning

- Training data are actually X and Y coming as pairs

$$\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{n_{\text{train}}}, \quad (x^{(i)}, y^{(i)}) \in \mathbb{X} \times \mathbb{Y}$$

- If \mathbb{Y} is continuous, then the task is called regression, and if \mathbb{Y} is discrete, then it is a classification problem.



$X =$

Example: Determine if an image encodes a cat or a dog
(called a **classification** task)

$$Y = \{1, 1, 0, 0\} \quad f_{\theta}(x^{(i)}) = \hat{y}^{(i)} \longrightarrow f_{\hat{\theta}}$$

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} R(\mathcal{D}, \theta)$$

Training data

Model

Optimisation



$x^{(i)}$ is the i th **data vector** of the training base, and $y^{(i)}$ is called the **target (or predicted) variable**

Supervised learning

- Training data are actually X and Y coming as pairs

$$\mathcal{D} = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^{n_{\text{train}}}, \quad (\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) \in \mathbb{X} \times \mathbb{Y}$$

- If \mathbb{Y} is continuous, then the task is called regression, and if \mathbb{Y} is discrete, then it is a classification problem.
- Ideally, we would like to minimise the **expected risk**, i.e. the **expected value of a loss function** $\ell(y, \hat{y})$

$$R(\mathcal{D}, \boldsymbol{\theta}) = \mathbb{E}_{X,y}[\ell(y, \hat{y})]$$

Loss function: measures how bad your model
is on a single example



However, we do not know $p(X, y)$ so in practice we rely on the **empirical risk** instead

$$\hat{R}(\mathcal{D}, \boldsymbol{\theta}) = \frac{1}{n_{\text{train}}} \sum_{i=1}^{n_{\text{train}}} \ell(y^{(i)}, \hat{y}^{(i)}).$$

Supervised learning

- Training data are actually X and Y coming as pairs

$$\mathcal{D} = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^{n_{\text{train}}}, \quad (\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) \in \mathbb{X} \times \mathbb{Y}$$

- If \mathbb{Y} is continuous, then the task is called regression, and if \mathbb{Y} is discrete, then it is a classification problem.

Examples of tasks

Classification

Regression

Timeseries prediction

Segmentation

Examples of models

Artificial Neural network

Random forest

Linear regression

Logistic regression

Naïve Bayes

Nearest neighbours

Unsupervised learning

- Training data are the set of $x^{(i)}$'s only; no known results to predict
- In unsupervised learning, one seeks **patterns or structures** in X without prior labels
- Usually boils down to model the probability distribution of the dataset



$X =$

$x^{(1)}$

$x^{(2)}$

$x^{(3)}$

$x^{(4)}$

Example: Generate new images of cats and dogs (called a **sampling** task)

$$f_{\theta}(x) = p_{\theta}(x)$$

$$f_{\hat{\theta}}$$

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} R(\mathcal{D}, \theta)$$

such that $p_{\theta}(x) \approx p(x)$

Training data

Model

Optimisation

Unsupervised learning

- Training data are the set of $x^{(i)}$'s only; no known results to predict
- In unsupervised learning, one seeks **patterns or structures** in X without prior labels
- Usually boils down to model the probability distribution of the dataset

Examples of tasks

Clustering

Data augmentation

Dimensionality reduction

Sampling

Examples of models

Autoencoder

Boltzmann Machine

Diffusion models

Gaussian mixture model

Generative Adversarial network

Families of ML problems

Introduction to ML

Linear models

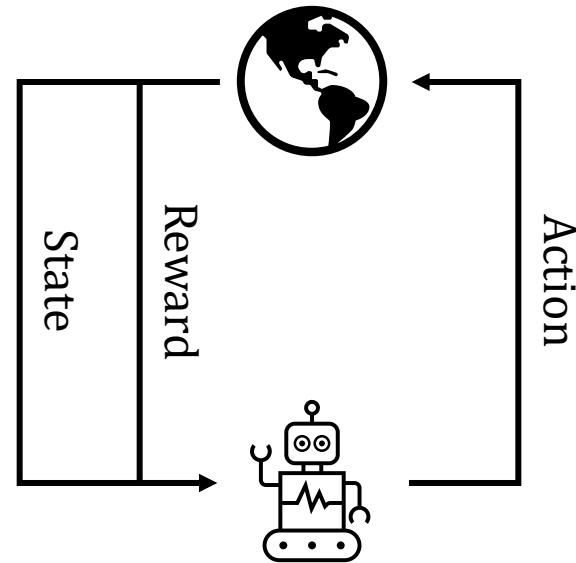
SL principles

Trees and neural networks

Risk optimisation

Reinforcement learning

- The philosophy is different: the model does not try to “imitate” like in supervised learning nor to find patterns but “tries” things
- It is based on an **agent** interacting with an **environment**
- The agent tries to find the best possible sequence of states and actions to **maximise a reward**



Examples of tasks

Game theory

Robotics

Autonomous driving

Examples of models

Markov decision processes

Q-networks

Deep policy gradient

Not discussed in this course, but a very good reference is [Reinforcement Learning – An introduction, Sutton and Barto, 2018](#)

Why do we need ML?

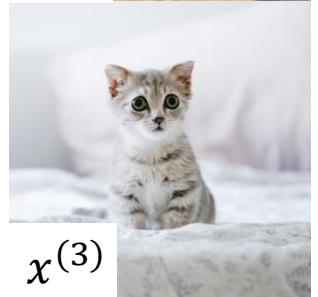
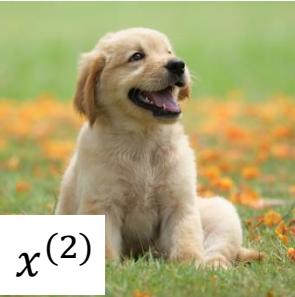
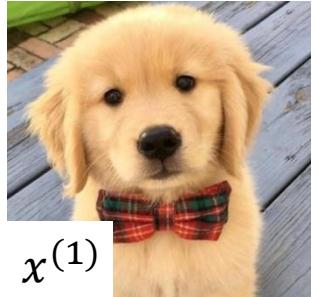
Introduction to ML

Linear models

SL principles

Trees and neural networks

Risk optimisation



$$d \approx 10^6$$

- To sample a $[0,1]^d$ space with a shortest distance to a test point at most ϵ , we need $n_{\text{train}} \geq \epsilon^{-d} = e^{-d \log \epsilon}$
- $d \approx 80$ requires more samples than the number of atoms in the universe**

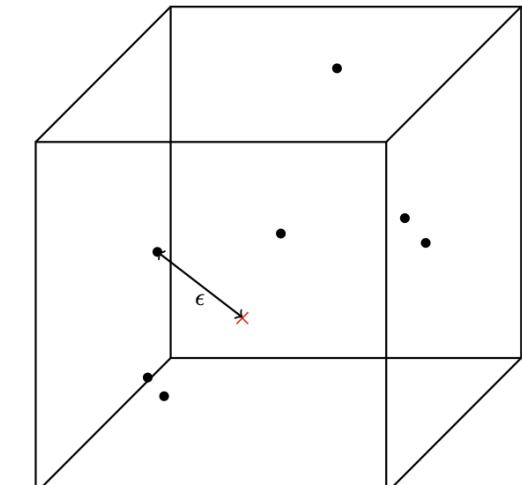
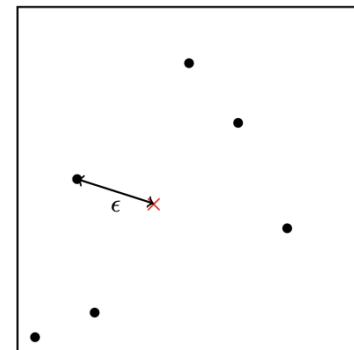
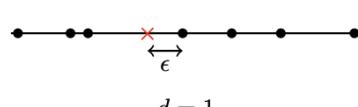
1-nearest neighbour

- A simple classification rule is for instance associating to a data the label of its closest neighbour in the d -dimensional space.

$$\hat{y} = y^{(m)} \text{ with } m = \operatorname{argmin}_i \|x - x^{(i)}\|_2^2$$

- In this case $R_{\text{train}} = 0$ but R_{test} is very large! Why?**

Curse of dimensionality





Traditional methods typically break down in high-dimensional spaces (**curse of dimensionality**) and it is impossible to design handcrafted decision rules for complex tasks.

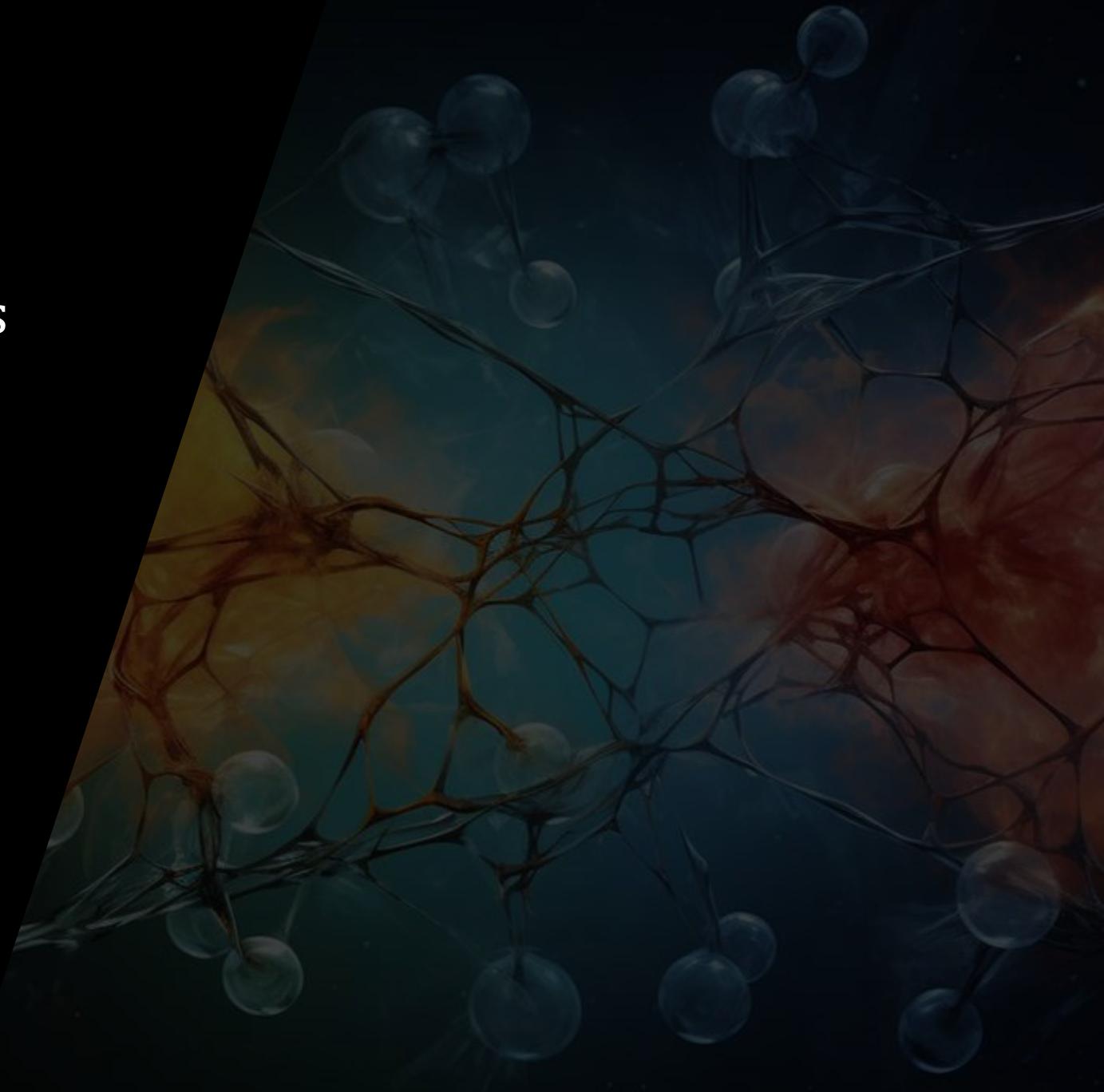
The **curse of dimensionality** is the **central problem of machine learning**. To fight it, ML relies on **prior information** about the problem:

- **Find an appropriate embedding** or feature representation of the data to simplify the problem,
- Exploit **structures** in the data (invariances, sparsity, long-range correlations, etc.) to define the model,
- **Penalise** complex models leading to poor generalisation performances using regularisation.

Linear models on feature vectors

Contents:

- *Linear regression model*
- *L2-loss for regression and normal equations*
- *Linear classification model*
- *Softmax function, cross-entropy loss for classification*



Linear regression

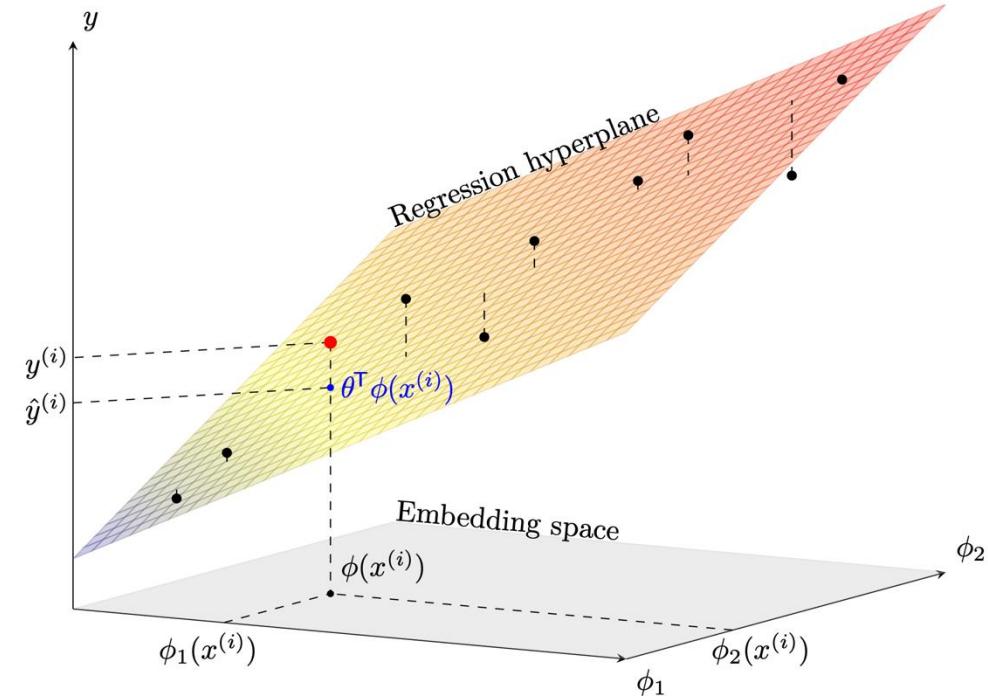
- What kind of problems one can solve efficiently, even in large dimensions? → **Linear systems!**
- Let us talk first about regression: the answer is modelled as

$$f_{\theta}(\phi_1^{(i)}, \phi_2^{(i)}, \dots) = \theta^T \phi^{(i)} = \hat{y}^{(i)}$$

- It is sometimes convenient to add an affine term (also called **bias** in the neural network literature), which can be absorbed in the feature vector making it of dimension $d' + 1$ where $\theta = [\theta_0, \theta_1, \theta_2, \dots]^T$, $\phi^{(i)} = [1, \phi_1^{(i)}, \phi_2^{(i)}, \dots]^T$.

$$\theta^T \phi(x^{(i)})$$

- **Geometric interpretation:** projection of an embedding vector onto a **hyperplane** parameterised by θ .



Linear regression

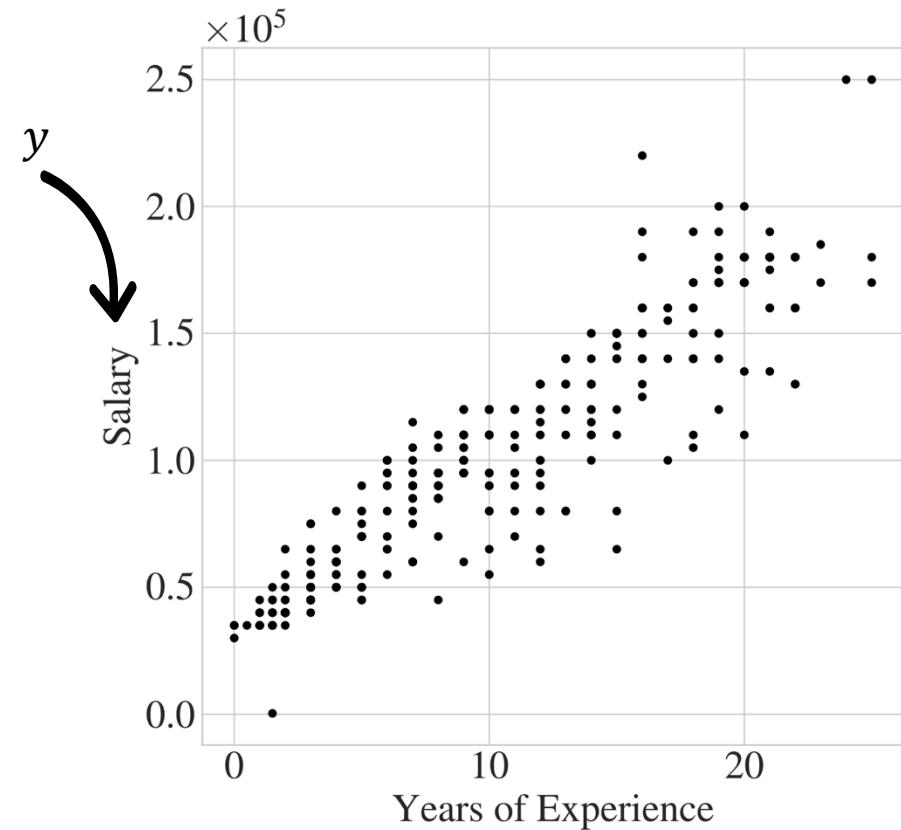
- Example: salary prediction based on the years of experience
- Data are $n = 373$ couples $(\phi^{(i)}, y^{(i)}) \Rightarrow$ **Supervised learning**
- The target variable $y \in \mathbb{R}$ is continuous \Rightarrow **Regression**
- The linear model is

$$\hat{y}^{(i)} = \theta_0 + \theta_1 \phi_1^{(i)},$$

where $\phi_1^{(i)}$ is the nb. of years of experience of the i^{th} training example

- Now the model is fixed, how to find $\hat{\theta}$, the best possible parameters for our model and data?
- This is done using **empirical risk minimisation (ERM)**

$$\hat{\theta} = \operatorname{argmin}_{\theta} R(\mathcal{D}, \theta) = \operatorname{argmin}_{\theta} \frac{1}{n_{\text{train}}} \sum_{i=1}^{n_{\text{train}}} \ell(\hat{y}^{(i)}, y^{(i)})$$



Linear regression: solution to ERM

Introduction to ML

Linear models

SL principles

Trees and neural networks

Risk optimisation

Linear regression

- A common **choice** of loss for regression is a **squared loss function**

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \frac{1}{n_{\text{train}}} \sum_{i=1}^{n_{\text{train}}} (\hat{y}^{(i)} - y^{(i)})^2$$

- Here**, the optimisation problem can be solved analytically in closed-form. Rewriting the risk matricially, we have

$$R(\mathbf{X}, \boldsymbol{\theta}) = \frac{1}{n_{\text{train}}} \|\Phi \boldsymbol{\theta} - \mathbf{y}\|_2^2$$

Feature matrix

$$\Phi = \begin{pmatrix} \phi_1^{(1)} & \dots & \phi_{d'}^{(1)} \\ \vdots & \ddots & \vdots \\ \phi_1^{(n_{\text{train}})} & \dots & \phi_{d'}^{(n_{\text{train}})} \end{pmatrix} \in \mathbb{R}^{n_{\text{train}} \times d'}$$

Target vector

$$\mathbf{y} = [y^{(1)}, \dots, y^{(n_{\text{train}})}]^T \in \mathbb{R}^{n_{\text{train}}}$$



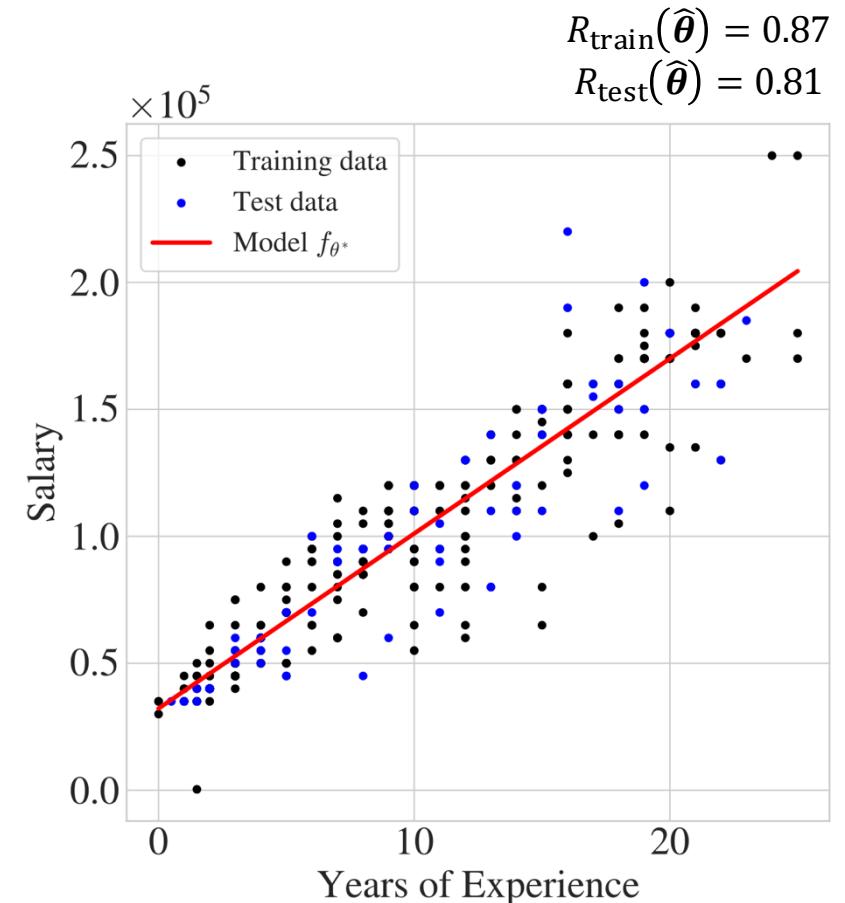
The analytical minimisation of the squared loss in linear regression gives the unique solution (when $d' < n$) known as **normal equations**

$$\hat{\boldsymbol{\theta}} = (\Phi^T \Phi)^{-1} \Phi^T \mathbf{y}$$

Linear regression



- I first separated the dataset into **training and test sets**, $n_{\text{train}} = 0.8n$ and $n_{\text{test}} = 0.2n$ chosen randomly.
 - Then, I computed the optimal parameters minimising the empirical risk using the normal equations on the training features
- $$\hat{\theta} = (\Phi^T \Phi)^{-1} \Phi^T y.$$
- I computed the risk on the train and test sets and found they are close.



Exactly solvable model, low variance



Cannot represent local relationships, may be biased

Linear regression

Remark

- The choice of the squared loss can also be motivated from a probabilistic point of view
- Assuming a Gaussian distribution for the error $\epsilon^{(i)} = \hat{y}^{(i)} - y^{(i)} \sim \mathcal{N}(0, \sigma^2)$ and **independent** observations, the **likelihood** can be written

$$p(X|\boldsymbol{\theta}) = \prod_i p(y^{(i)}|x^{(i)}, \boldsymbol{\theta})$$

- Maximising the log-likelihood to obtain the parameters of the model gives

$$\max_{\boldsymbol{\theta}} \log p(X|\boldsymbol{\theta}) = \max_{\boldsymbol{\theta}} -\frac{1}{2\sigma_\epsilon^2} \sum_i (y^{(i)} - \hat{y}^{(i)})^2$$

→ **The maximum likelihood estimator (MLE) is the same as the empirical risk minimiser under a squared loss function to measure the error of the model**

Linear classification: the model

Introduction to ML

Linear models

SL principles

Trees and neural networks

Risk optimisation

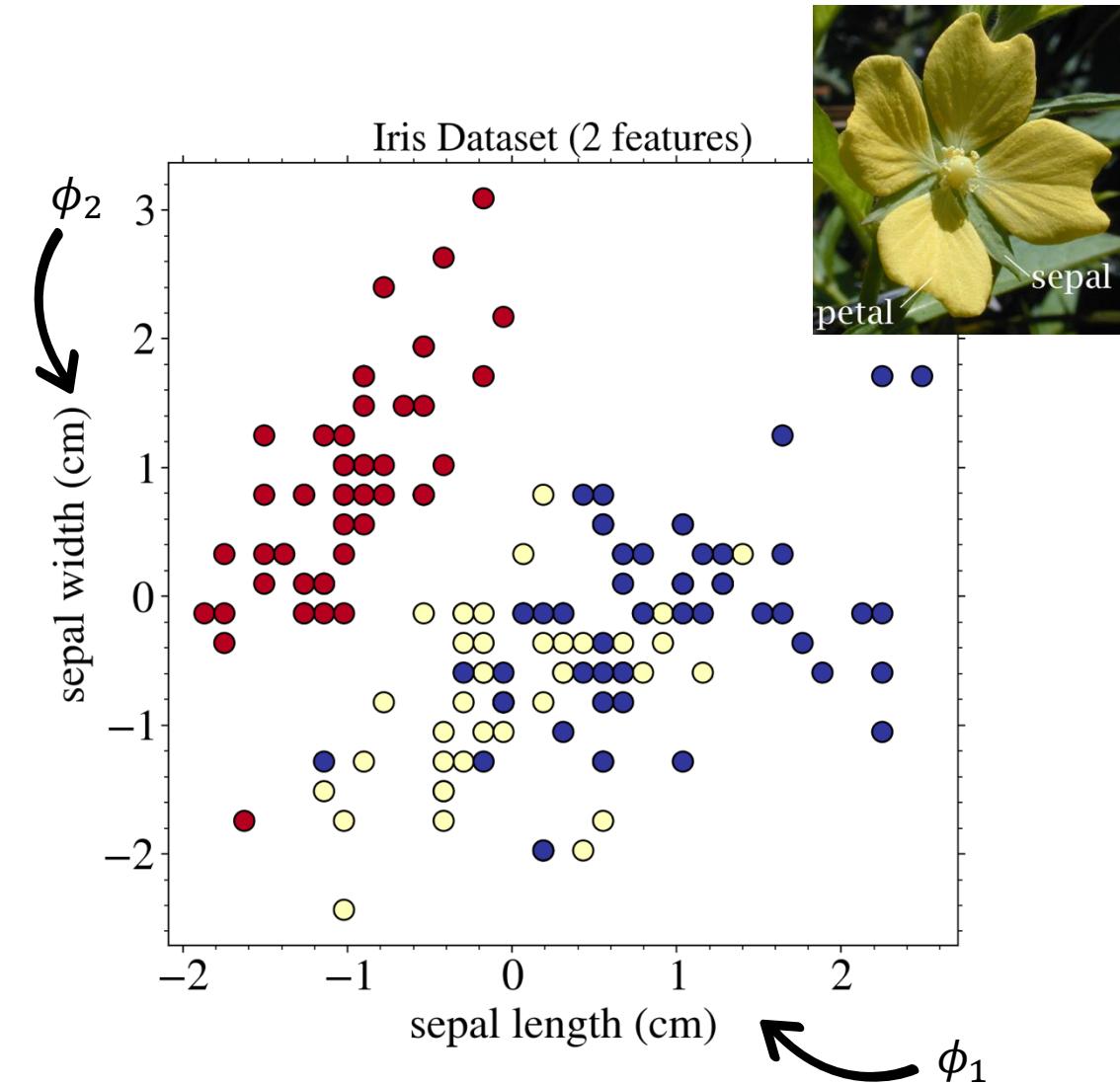
Linear classification

- Consider a K -class classification problem for which features ϕ allow linear separability
- Example: Iris dataset with $n = 150$ couples $(\phi^{(i)}, y^{(i)}) \Rightarrow$ **Supervised learning**
- The target variable $y \in \{0,1,2\} \Rightarrow$ **Classification**
- A natural loss function for classification is the **0-1 loss**

$$\ell(y, \hat{y}) = \begin{cases} 1 & \text{if } \hat{y}^{(i)} \neq y^{(i)}, \\ 0 & \text{otherwise.} \end{cases}$$

- The optimal decision rule (in Bayes sense) is

$$\hat{y} = \operatorname{argmax}_k p(y = k | \phi).$$



We thus need a **model** $p_\theta(y = k | \phi)$ of the conditional probability distribution to perform classification!



Linear classification: the model

Introduction to ML

Linear models

SL principles

Trees and neural networks

Risk optimisation

Linear classification

- The simplest models assume a linear log probability

$$\log p_\theta(y^{(i)} = k | \phi^{(i)}) = \theta_k^T \phi^{(i)} - \log Z$$

where Z is a normalizing constant so that probabilities sum to one.

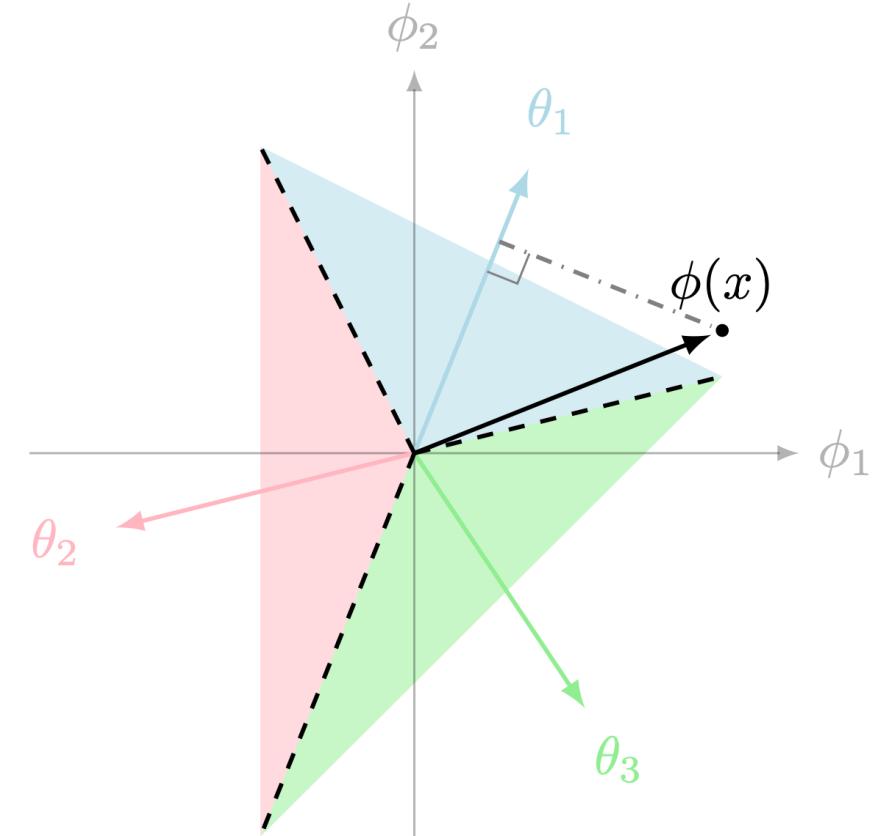
- It means that

$$p_\theta(y^{(i)} = k | \phi^{(i)}) = \frac{\exp(\theta_k^T \phi^{(i)})}{\sum_{j=1}^K \exp(\theta_j^T \phi^{(i)})}$$

which is called the **softmax function** allowing to turn the linear responses for each class into probabilities.

- And the classification rule is

$$\hat{y} = \operatorname{argmax}_k \theta_k^T \phi^{(i)}$$



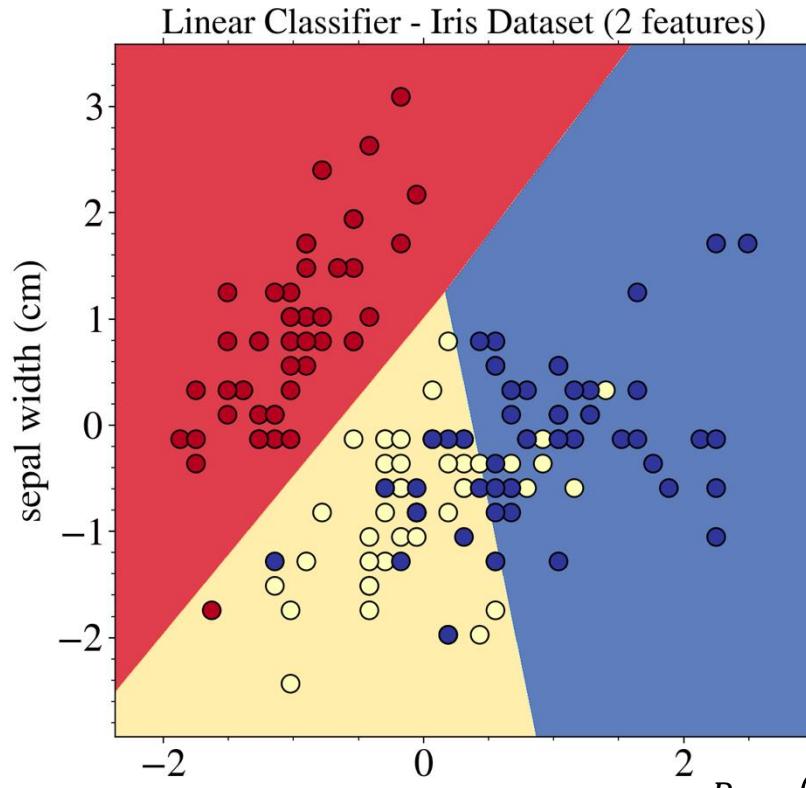
Geometrically, it corresponds to computing the overlap between the feature $\phi^{(i)}$ and a vector representative for each class, θ_k , and associating **the class maximising the dot product**, leading to **linear decision** boundaries shown as hyperplanes.

Linear classification

- Now the model is specified, we need minimise the risk to obtain the parameters θ_k using some training data
- For optimisation, the 0-1 loss is not suitable since it is not differentiable, but we can relax it using the probabilities

$$R(\mathcal{D}, \boldsymbol{\theta}) = - \sum_{i=1}^{n_{\text{train}}} \sum_{k=1}^K 1_{y^{(i)}=k} \log p_{\boldsymbol{\theta}}(y^{(i)} = k | \boldsymbol{\phi}^{(i)})$$

which is **now differentiable and convex**.



$$R_{\text{train}}(\hat{\boldsymbol{\theta}}) = 0.38$$

$$R_{\text{test}}(\hat{\boldsymbol{\theta}}) = 0.48$$



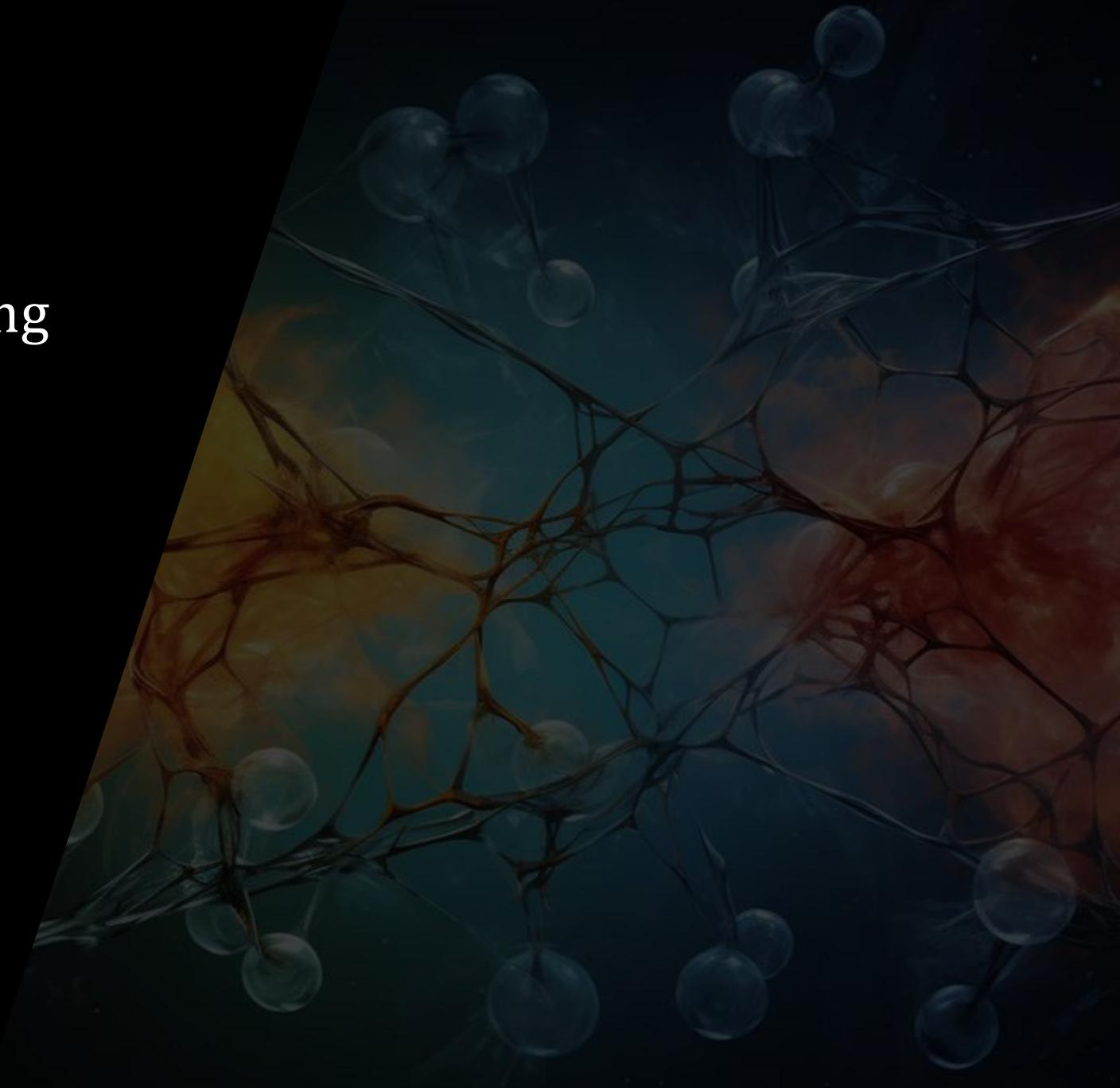
This risk is referred to as ***cross-entropy*** and it is the most widely used cost function for classification problems. The parameters of the model are then obtained by minimising the risk, i.e.

$$\hat{\boldsymbol{\theta}} = \operatorname{argmin}_{\boldsymbol{\theta}} R(\mathcal{D}, \boldsymbol{\theta}).$$

Principles of Supervised Learning

Contents:

- *Bias-variance trade-off for supervised problems*
- *Overfitting, underfitting and test set*
- *Explicit regularisation*



In the previous chapter, we have:

1. Specified different models for different supervised learning tasks (regression and classification),
2. Specified loss functions and associated empirical risks,
3. Used a finite training set to minimise the empirical risk.
4. Found parameters of our models $f_{\theta}(\phi^{(i)})$

Now what could possibly go wrong with our models?

Generalisation!

Is it able to work on new, independent from training, data?

The bias-variance trade-off

Introduction to ML

Linear models

SL principles

Trees and neural networks

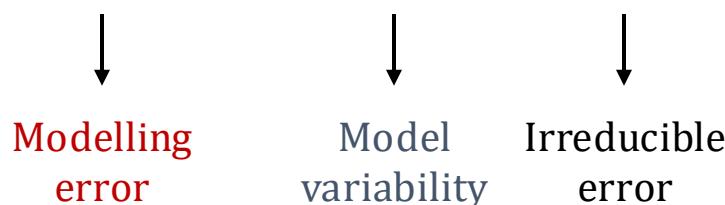
Risk optimisation

- The whole aim of training supervised models is to generalise well on unseen data. In practice, we would like to minimise $\mathbb{E}_{x,y}(\ell(f_{\theta}(x), y))$ that we approximate by $\frac{1}{n_{\text{train}}} \sum_{i=1}^{n_{\text{train}}} \ell((f_{\theta}(x^{(i)}), y^{(i)}))$
- In a regression context, suppose there exists f such that $y^{(i)} = f(x^{(i)}) + \epsilon^{(i)}$, with $\mathbb{E}[\epsilon^{(i)}] = 0, \mathbb{E}[\epsilon^{(i)2}] = \sigma_{\epsilon}^2$
- We build a model f_{θ} of f minimising the squared error $\ell((f_{\theta}(x^{(i)}), y^{(i)})) = (y^{(i)} - f_{\theta}(x^{(i)}))^2$
- We can show that the expected risk on a test example \tilde{x} decomposes as

$$\mathbb{E}[(y - f_{\theta}(\tilde{x}))^2] = \mathbb{E}[f_{\theta}(\tilde{x}) - f(\tilde{x})]^2 + \mathbb{E}[(f_{\theta}(\tilde{x}) - \mathbb{E}[f_{\theta}(\tilde{x})])^2] + \sigma_{\epsilon}^2$$

$$\mathbb{E}[(y - f_{\theta}(\tilde{x}))^2] = \text{Bias}[f_{\theta}(\tilde{x})]^2 + \text{Var}[f_{\theta}(\tilde{x})] + \sigma_{\epsilon}^2$$

Note: Every terms are >0



Note: a similar expression holds for classification

Model complexity and generalisation error

31

Introduction to ML

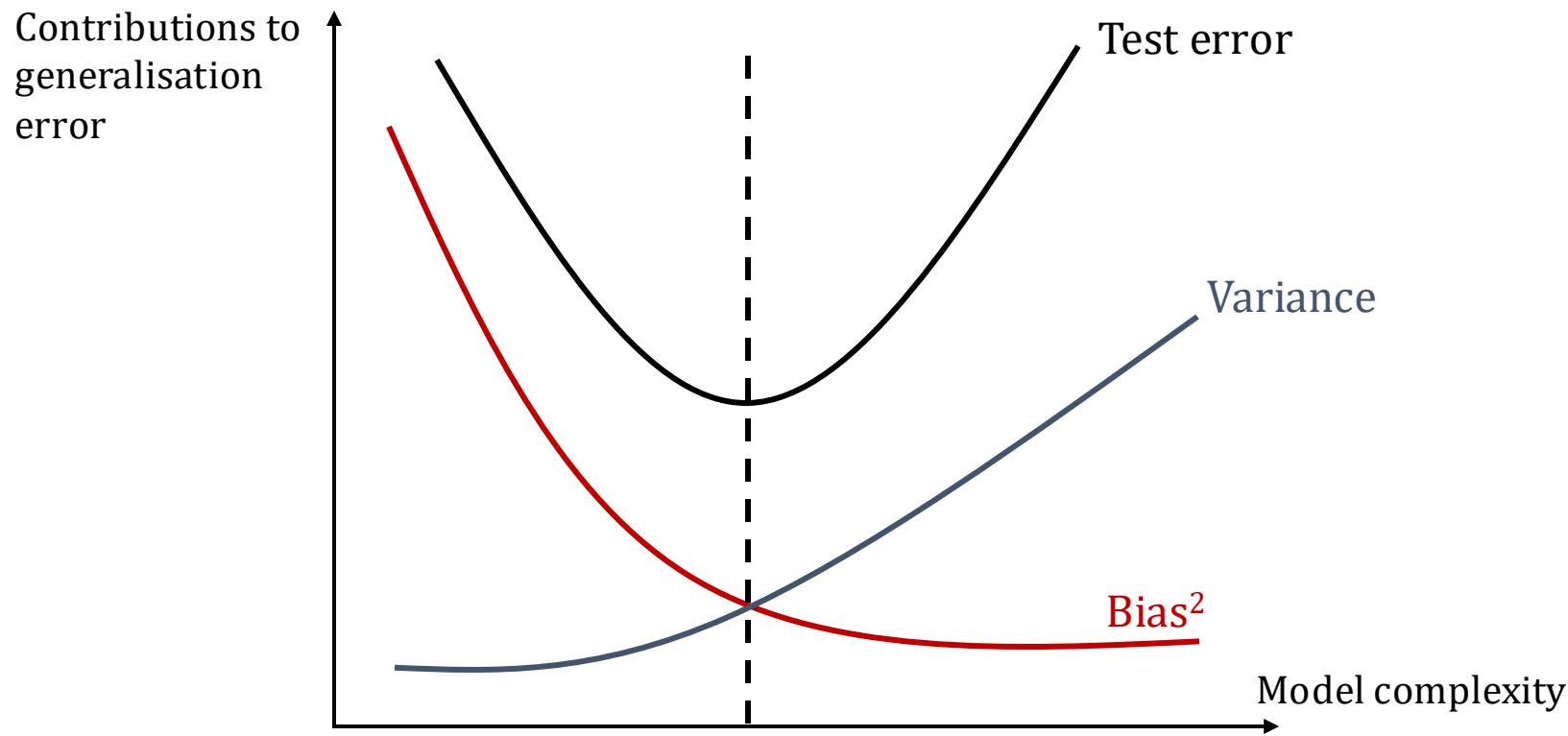
Linear models

SL principles

Trees and neural networks

Risk optimisation

- “Simple” models have **large bias** because they constrain very much the function class that is therefore far from the truth, but they usually have **low variance** and are robust to variations of the training set
- “Complex” models (with a lot of parameters for instance) have **small bias** but **large variance**



Model complexity and generalisation error

32

Introduction to ML

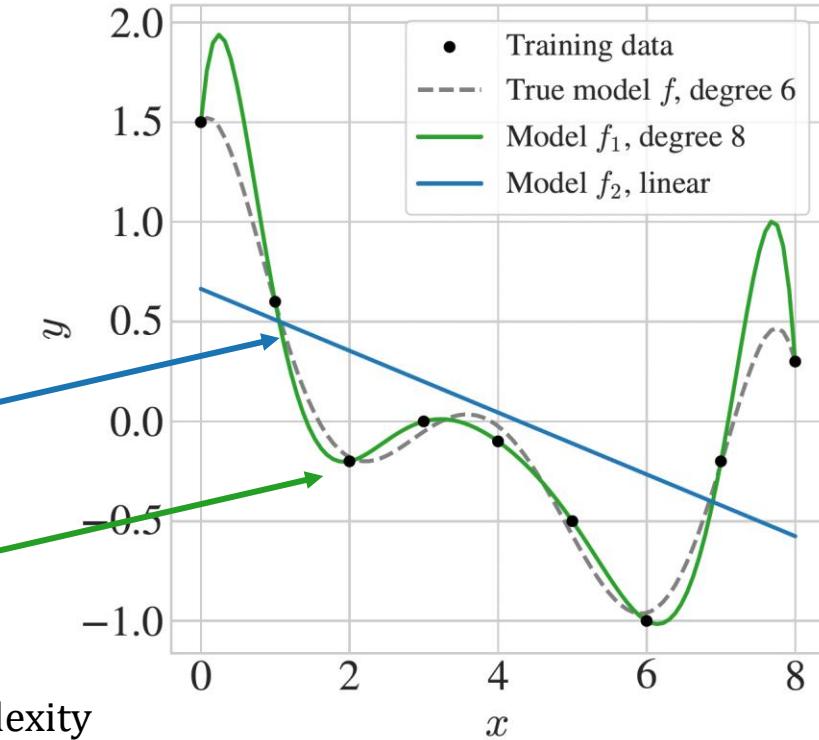
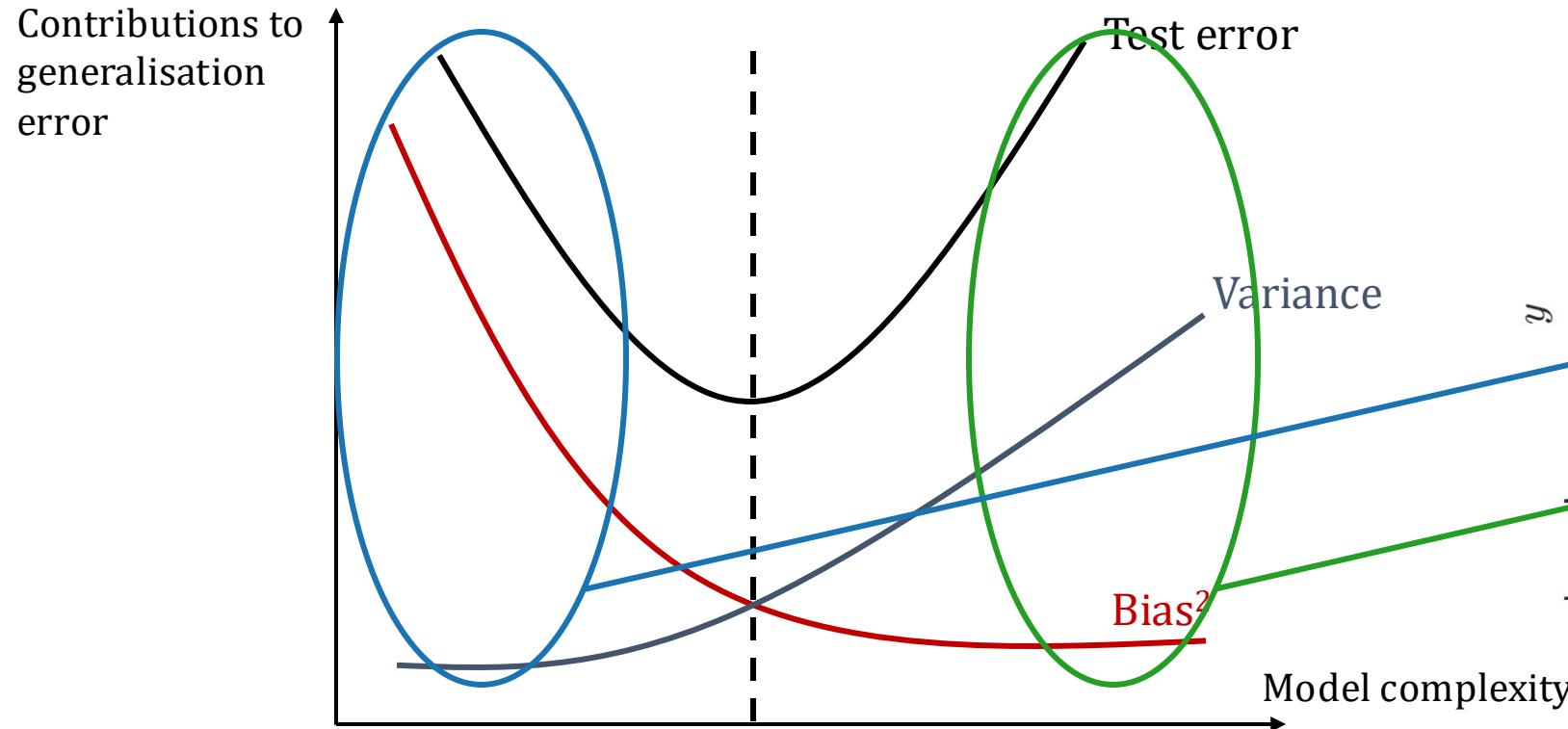
Linear models

SL principles

Trees and neural networks

Risk optimisation

- “Simple” models have **large bias** because they constrain very much the function class that is therefore far from the truth, but they usually have **low variance** and are robust to variations of the training set
- “Complex” models (with a lot of parameters for instance) have **small bias** but **large variance**



Model complexity and generalisation error

33

Introduction to ML

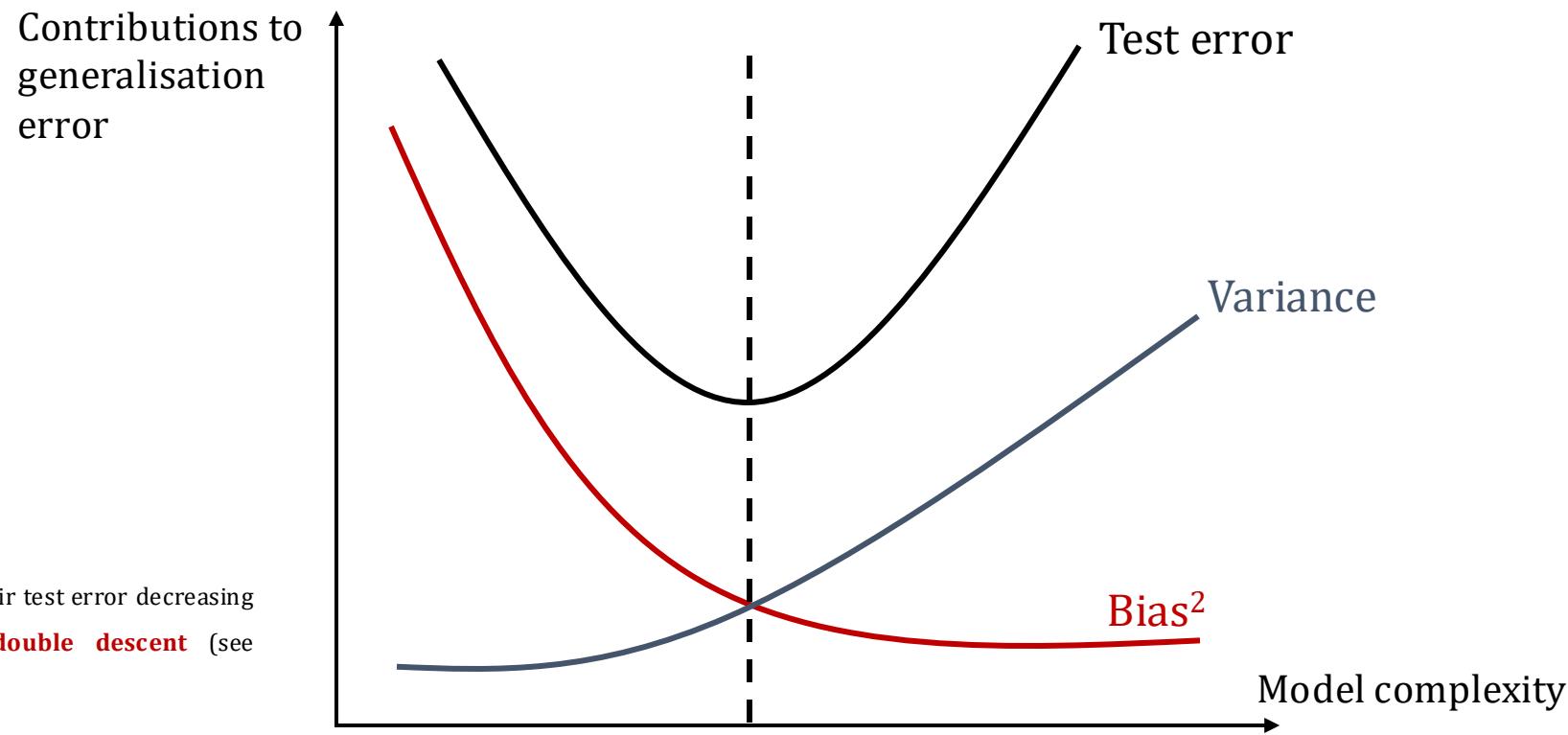
Linear models

SL principles

Trees and neural networks

Risk optimisation

- “Simple” models have **large bias** because they constrain very much the function class that is therefore far from the truth, but they usually have **low variance** and are robust to variations of the training set
- “Complex” models (with a lot of parameters for instance) have **small bias** but **large variance***



* Heavily overparametrized have their test error decreasing again, a phenomenon dubbed **double descent** (see [Belkin+18](#) and [Nakkiran+19](#))



Models need to be built such that they are not too flexible to fit the noise in the data but also not too restrictive to avoid bias

Model complexity and generalisation error

34

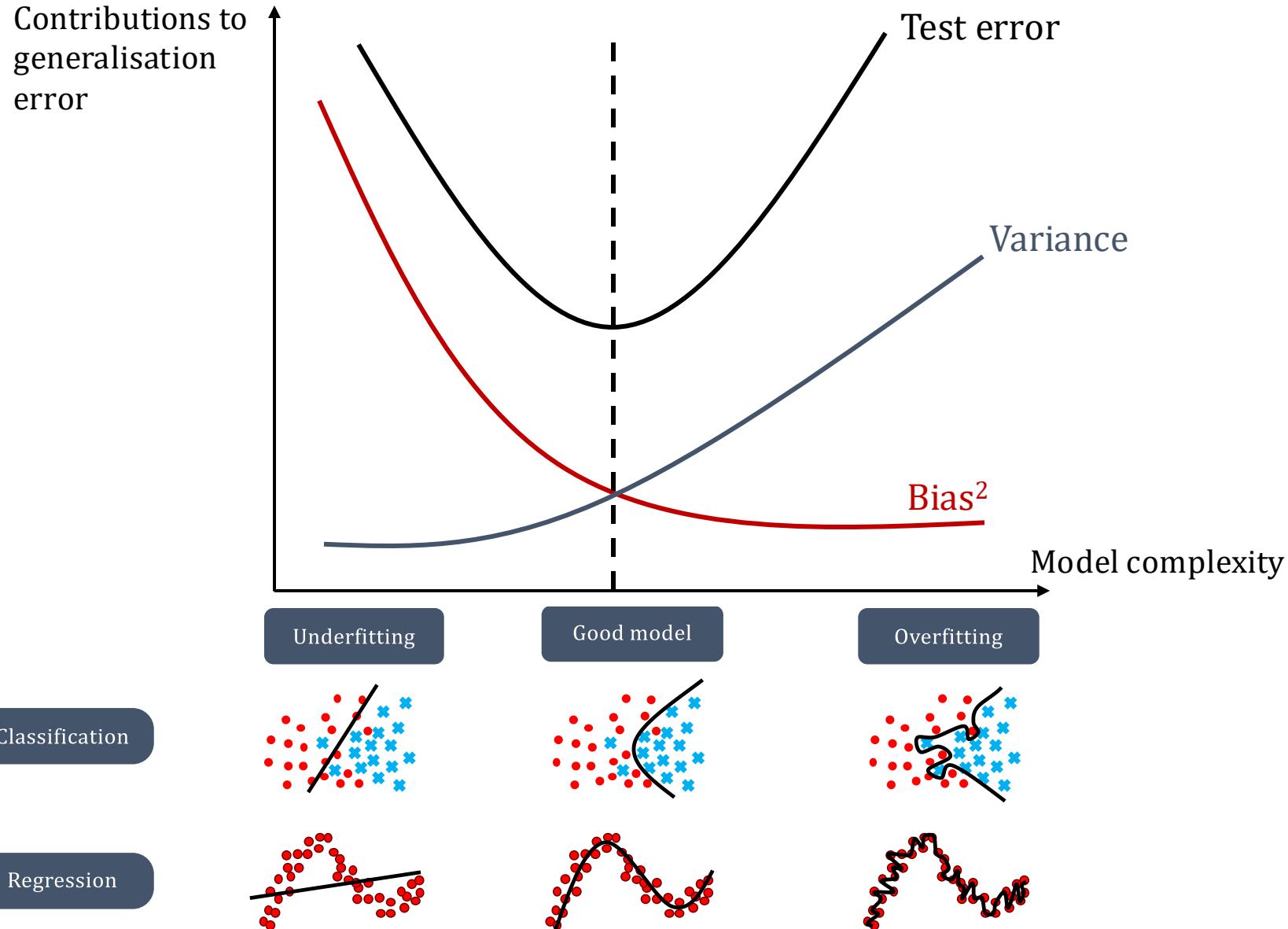
Introduction to ML

Linear models

SL principles

Trees and neural networks

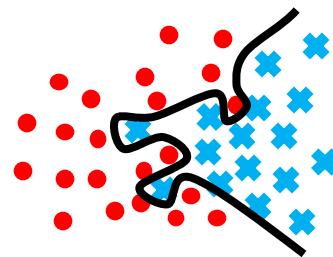
Risk optimisation



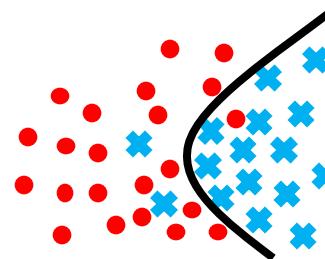
- To measure if we really learnt something useful in supervised learning in practice, we use a **test dataset** that the model has never seen but for which we know the labels and check that $R_{\text{train}}(\hat{\theta})$ and $R_{\text{test}}(\hat{\theta})$ are of the same order
- Based on the previous view, there are two regimes where things could go wrong: high variance and low bias models vs high bias and low variance models, respectively defining **overfitting** and **underfitting**

Classification

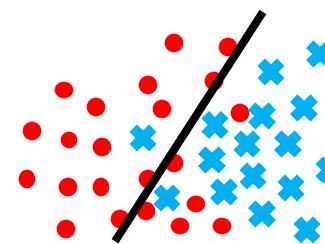
Overfitting



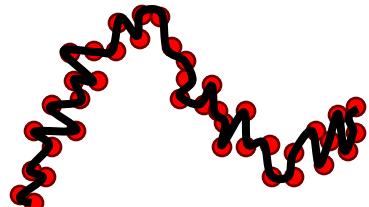
Good model



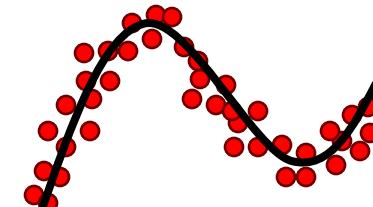
Underfitting



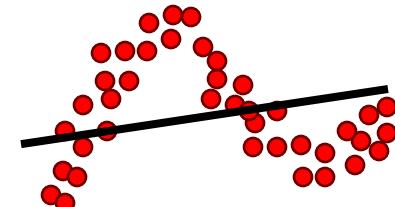
Regression



$$R_{\text{train}}(\hat{\theta}) \ll R_{\text{test}}(\hat{\theta})$$



$$R_{\text{train}}(\hat{\theta}) \approx R_{\text{test}}(\hat{\theta})$$



$$R_{\text{train}}(\hat{\theta}) \approx R_{\text{test}}(\hat{\theta}) \text{ but large}$$

Some fixes to under and over fitting

36

Introduction to ML

Linear models

SL principles

Trees and neural networks

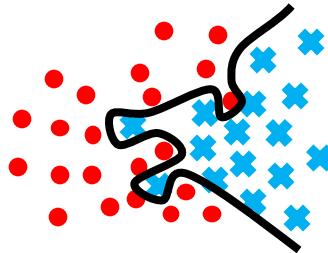
Risk optimisation

Overfitting

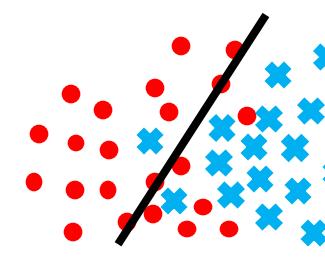
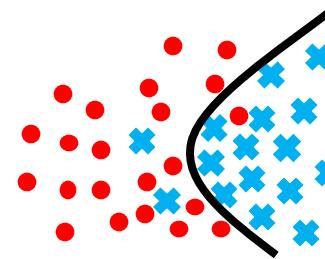
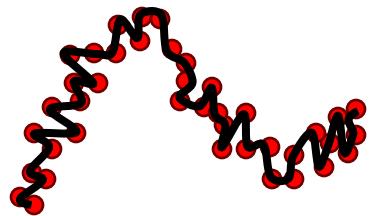
Good model

Underfitting

Classification



Regression



Possible fixes

- Add more data
- Remove features
- Stop the training earlier
- Add regularisation

- You did a good job!

$R_{\text{train}}(\hat{\theta}) \approx R_{\text{test}}(\hat{\theta})$
but large

- Try a more complex model
- Train your model longer

Overfitting in linear model

37

Introduction to ML

Linear models

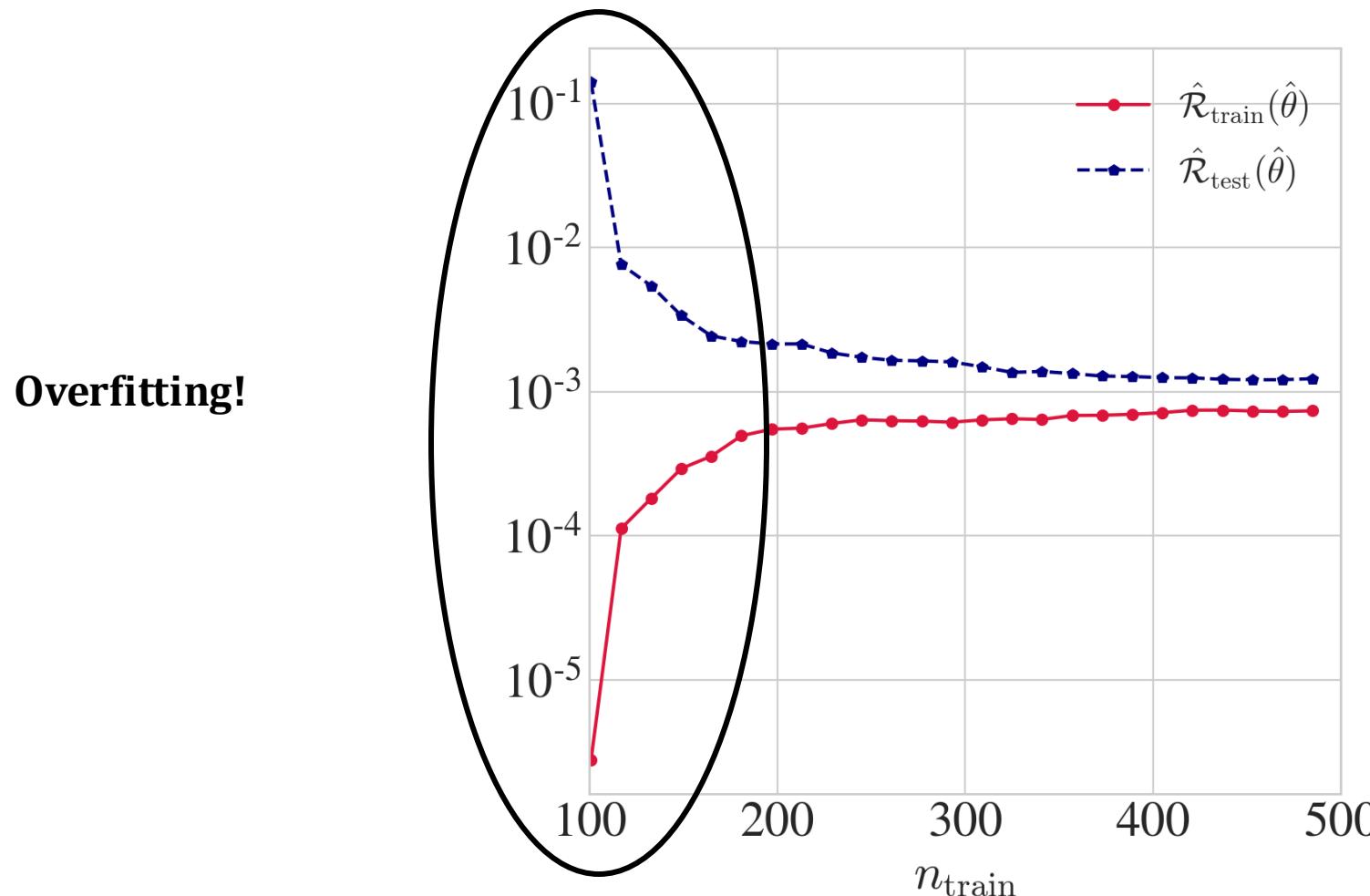
SL principles

Trees and neural networks

Risk optimisation

- Back to our regression problem: what happens when $n_{\text{train}} \approx d'$? Remember the solution

$$\hat{\theta} = (\Phi^T \Phi)^{-1} \Phi^T y$$



Linear regression with $d' = 100$ features

A gentle introduction to regularisation

38

Introduction to ML

Linear models

SL principles

Trees and neural networks

Risk optimisation

- One generic way to deal with **overfitting** is by penalising ‘complex’ models and restrain the class of learned functions: this is called explicit **regularization** and appears in the optimization problem as a constraint on parameter space

$$\hat{\theta} = \operatorname{argmin}_{\theta \in \Theta} R(\theta)$$

Unregularized
optimization

$$\hat{\theta} = \operatorname{argmin}_{\theta \in \Theta} R(\theta) \text{ s. t. } P(\theta) \leq \epsilon$$

Regularized
optimization

- The function $P(\theta)$ is called **penalty function** and the regularized problem can in fact be written equivalently as (by Lagrange duality)

$$\hat{\theta} = \operatorname{argmin}_{\theta \in \Theta} R(\theta) + \lambda P(\theta)$$

1. λ is a **regularization** (hyper)**parameter**
2. $P(\theta)$ can take different forms depending on the penalty we chose to impose on the set of parameters

- Common penalty functions are the L_p -norms with $p = 1$ or 2

$$P(\theta) = \|\theta\|_p$$

Regularisation as a smoothness constraint

39

Introduction to ML

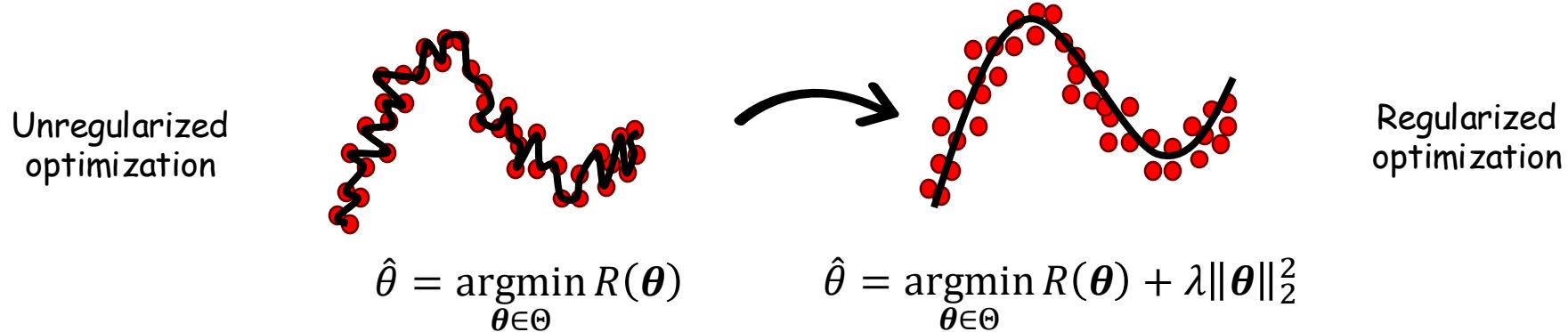
Linear models

SL principles

Trees and neural networks

Risk optimisation

- By solving the regularized optimization with **an appropriate value of λ** , we can reduce the overfitting



- Let us apply L_2 regularisation to our linear regression model which now minimises the regularised risk

$$\hat{\theta} = \operatorname{argmin}_{\theta \in \Theta} \frac{1}{n_{\text{train}}} \|\Phi\theta - Y\|_2^2 + \lambda \|\theta\|_2^2$$

- This can be minimised analytically and gives

$$\hat{\theta} = \frac{1}{n_{\text{train}}} \left(\frac{\Phi^T \Phi}{n_{\text{train}}} + \lambda I \right)^{-1} \Phi^T Y$$

where the matrix is now invertible for $\lambda > 0$.

Regularisation as a smoothness constraint

40

Introduction to ML

Linear models

SL principles

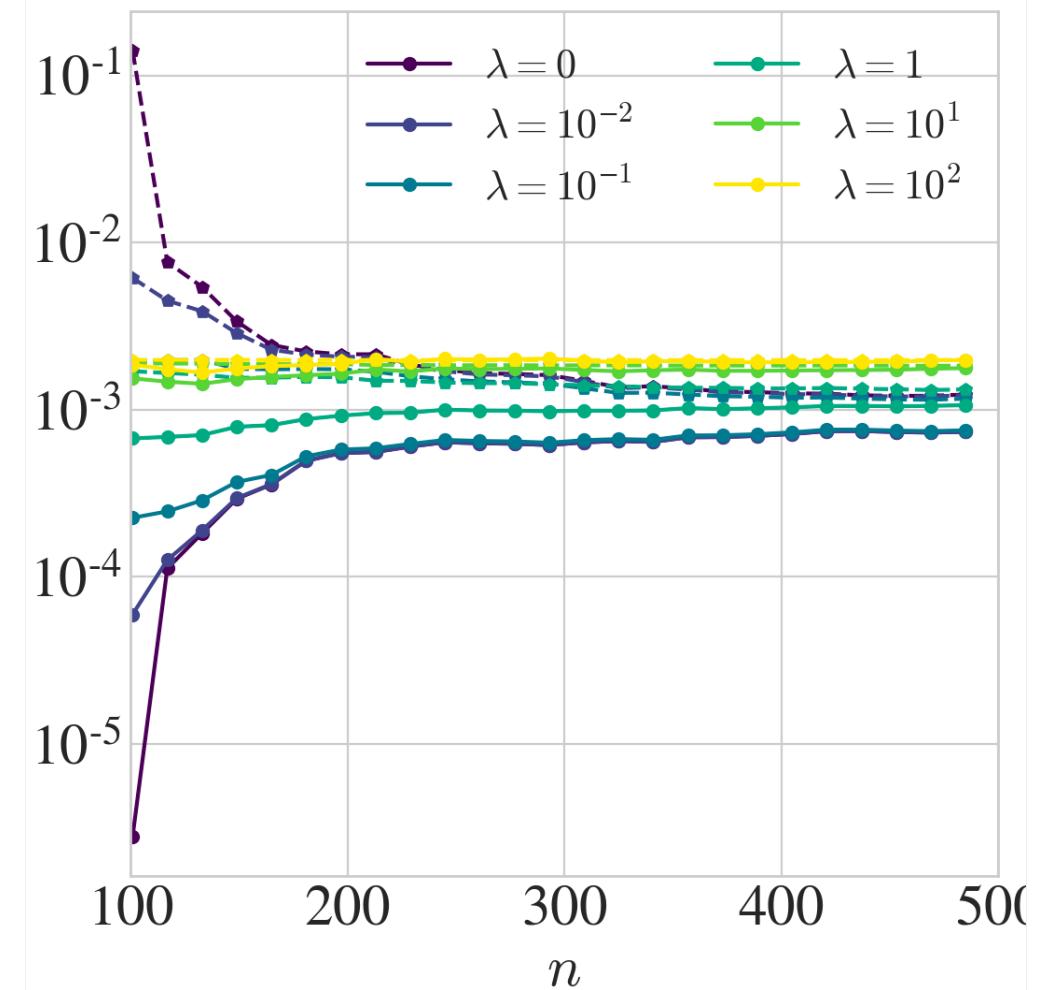
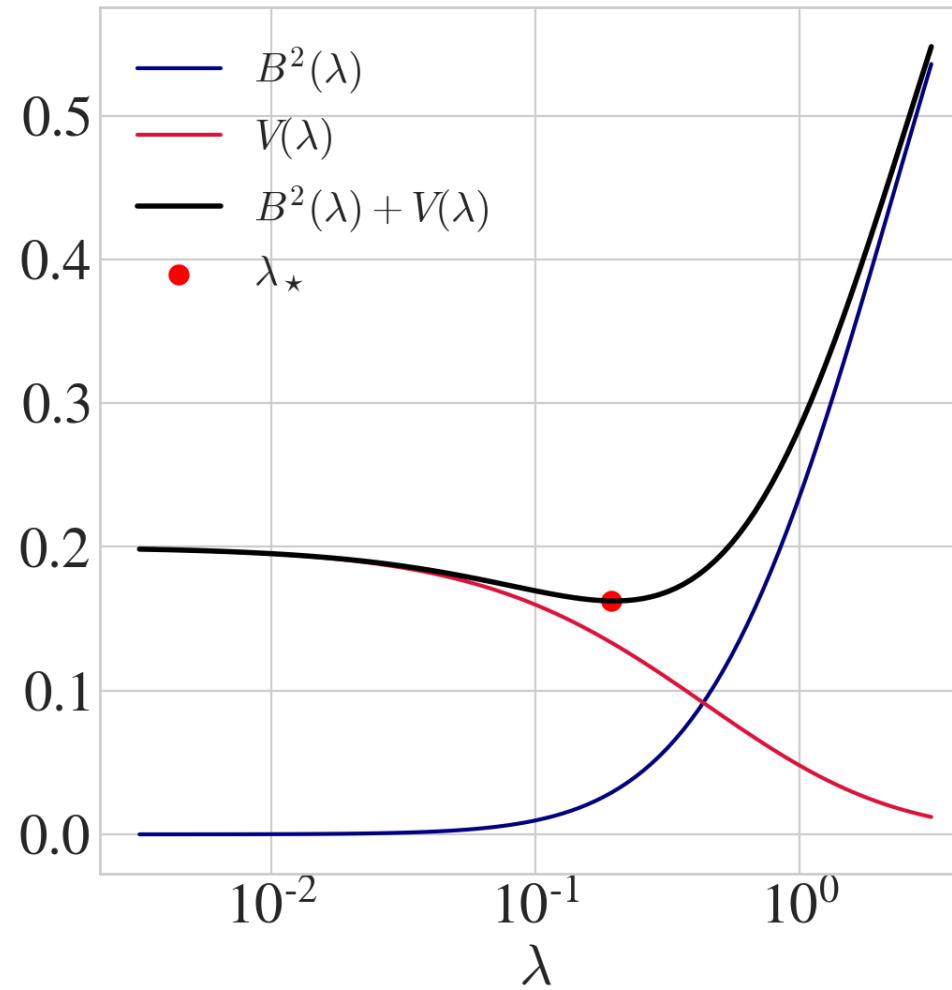
Trees and neural networks

Risk optimisation



The penalty impact the generalisation error by increasing the bias and decreasing the variance with λ

→ Regularisation helps reducing overfitting and improves the generalisation performances!



A word about hyperparameters

Introduction to ML

Linear models

SL principles

Trees and neural networks

Risk optimisation

- One **hyperparameter**: the regularisation parameter λ



Hyperparameter: parameter that is **not learned** during the optimisation.

Examples include regularisation parameter, depth of trees, learning rate in optimisation.

- For the regularisation parameter: a value that is too large introduces a large bias, while if too small and close to zero, we do not solve the overfitting issue
- Usually, we use **grid search** to find the hyperparameter performing best on a third dataset: the **validation set**



- **Training set:** training data used to learn parameters of the model during optimisation,
- **Test set:** independent set used to evaluate and compare the models (should in principle be used once by a model)
- **Validation set:** used to fit hyperparameters of the model by varying it and keeping the value minimising the validation error R_{valid} .

A word about hyperparameters

Introduction to ML

Linear models

SL principles

Trees and neural networks

Risk optimisation

- One **hyperparameter**: the regularisation parameter λ



Hyperparameter: parameter that is **not learned** during the optimisation.

Examples include regularisation parameter, depth of trees, learning rate in optimisation.

- For the regularisation parameter: a value that is too large introduces a large bias, while if too small and close to zero, we do not solve the overfitting issue
- Usually, we use **grid search** to find the hyperparameter performing best on a third dataset: the validation set or **cross-validation**

Cross-validation: Split the available data into k folds and train the model k times changing the chunk of validation set. At the end, average the obtained errors.



Credit: F. Bach

- Let us consider that $y^{(i)} = f_{\theta}(x^{(i)}) + \epsilon^{(i)}$, meaning our model is correct up to an error with $\epsilon^{(i)}$ that we assume i.i.d.
- In this, case we can write the **likelihood** of the data as

$$p(Y|X, \theta) = \prod_i p(y^{(i)}|x^{(i)}, \theta)$$

- Using the **Bayes theorem**, we can express the posterior probability distribution of the parameters given the dataset as

$$p(\theta|X) \propto p(Y|X, \theta)p(\theta)$$

- Assuming a Gaussian likelihood $\epsilon^{(i)} \sim \mathcal{N}(0, \sigma_{\epsilon}^2)$, $\theta_i \sim \mathcal{N}(0, \sigma_{\theta}^2)$, and taking the log of this expression yields

$$\log p(\theta|X) = -\frac{1}{2\sigma_{\epsilon}^2} \sum_i (y^{(i)} - \hat{y}^{(i)})^2 - \frac{1}{2\sigma_{\theta}^2} \sum_i \theta_i^2 + \text{cst.}$$

- Finally,

$$\max_{\theta} \log p(\theta|X) \Leftrightarrow \min_{\theta} \sum_i (y^{(i)} - \hat{y}^{(i)})^2 + \lambda \|\theta\|^2, \quad \text{with } \lambda = \sigma_{\epsilon}^2 / \sigma_{\theta}^2$$

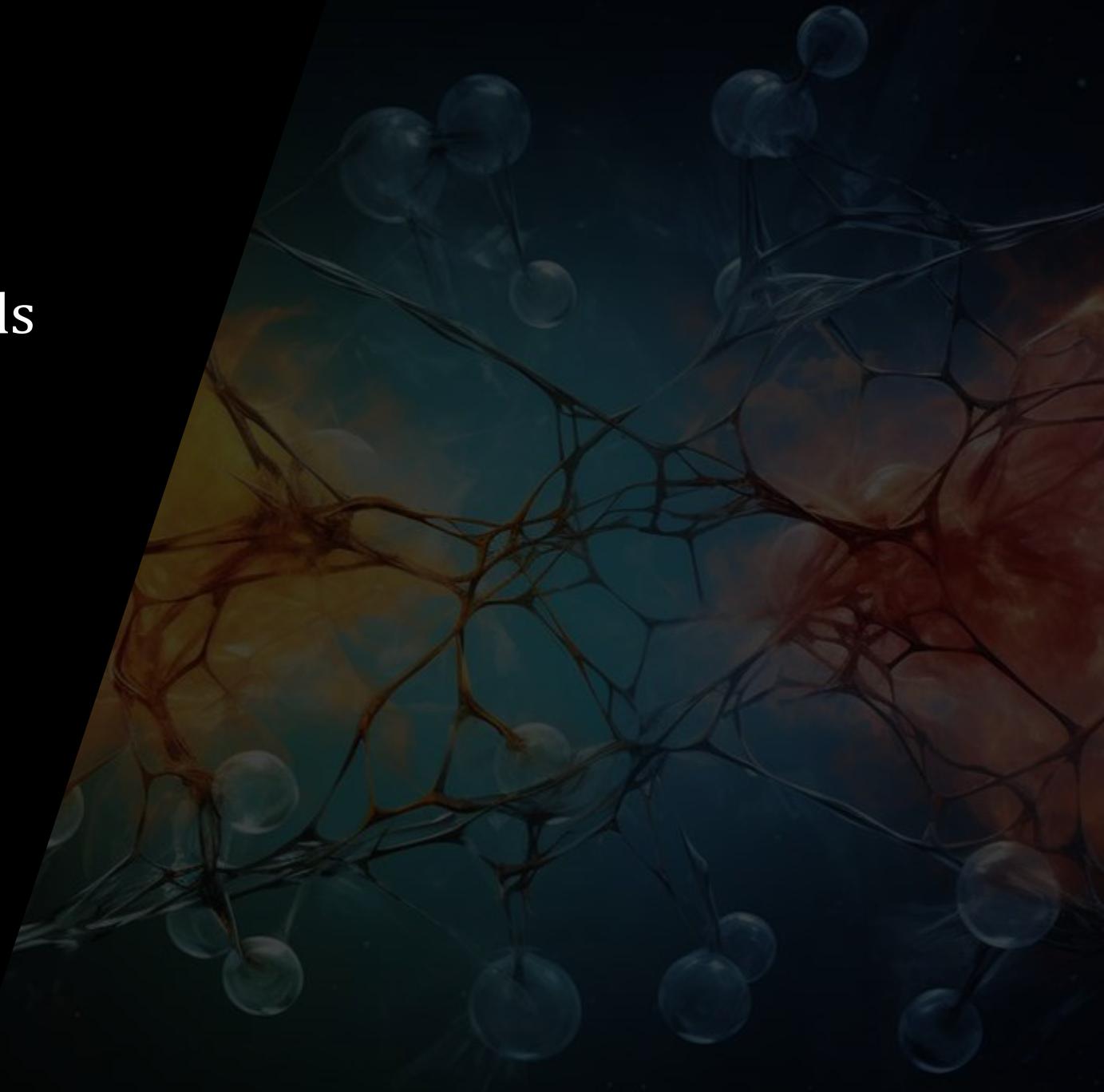


The maximum a posteriori estimator under Gaussian likelihood and Gaussian prior is equivalent to minimizing the square loss with an L_2 penalty. The **prior plays the role of the regularization**, and the **square-loss comes from the Gaussian error** assumption.

Other supervised learning models

Contents:

- *A first non-linear model: decision trees*
- *Ensembling: bagging and boosting*
- *Random forest and boosted trees*
- *Feed-forward neural networks*



A classification task: the XOR dataset

45

Introduction to ML

Linear models

SL principles

Trees and neural networks

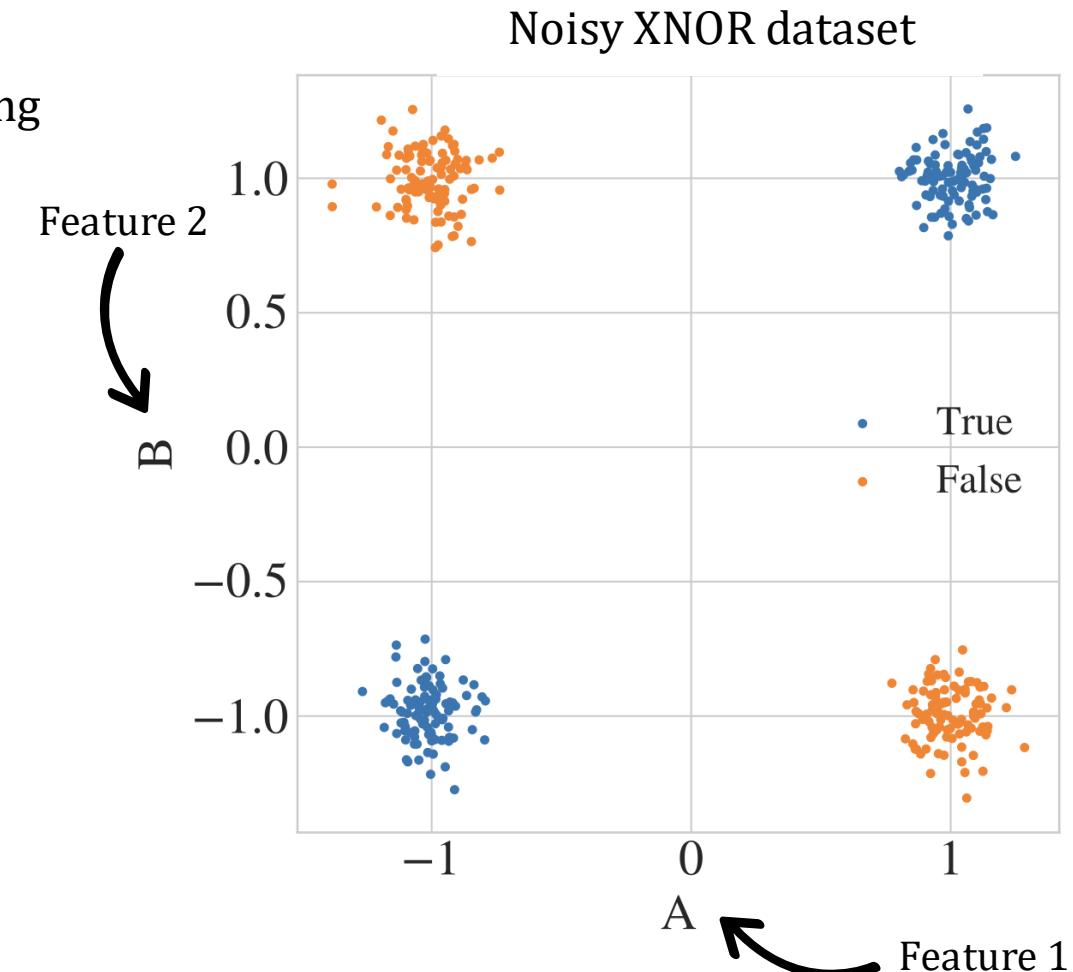
Risk optimisation

Decision trees

- Consider a **classification task** on an artificial dataset replicating the XNOR function

A	B	XNOR
True	True	True
True	False	False
False	True	False
False	False	True

- Data are $n = 500$ couples $(\phi^{(i)}, y^{(i)}) \Rightarrow$ **Supervised learning**
- The target variable $y \in \{-1, 1\}$ is discrete \Rightarrow **Classification**
- A linear classification would not be able to learn such a function*
- Decision trees** to the rescue!



*In fact, an alternative (not discussed here) would be to use **kernels** to find a higher dimensional space in which the data separates linearly and then use a linear classifier.

Intuition behind decision trees

Introduction to ML

Linear models

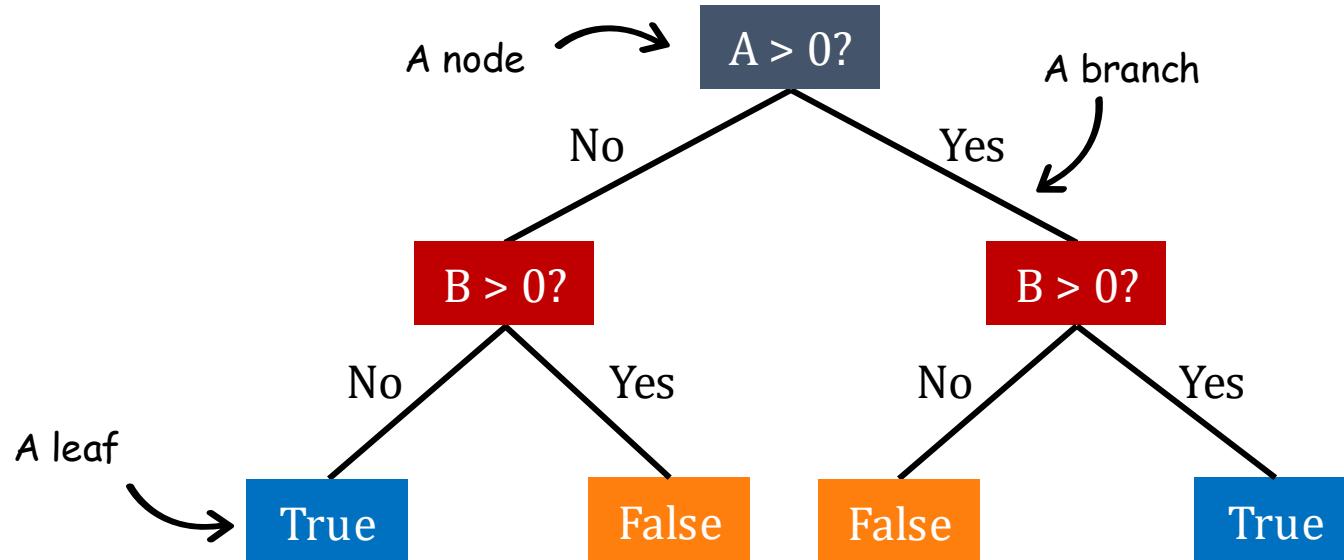
SL principles

Trees and neural networks

Risk optimisation

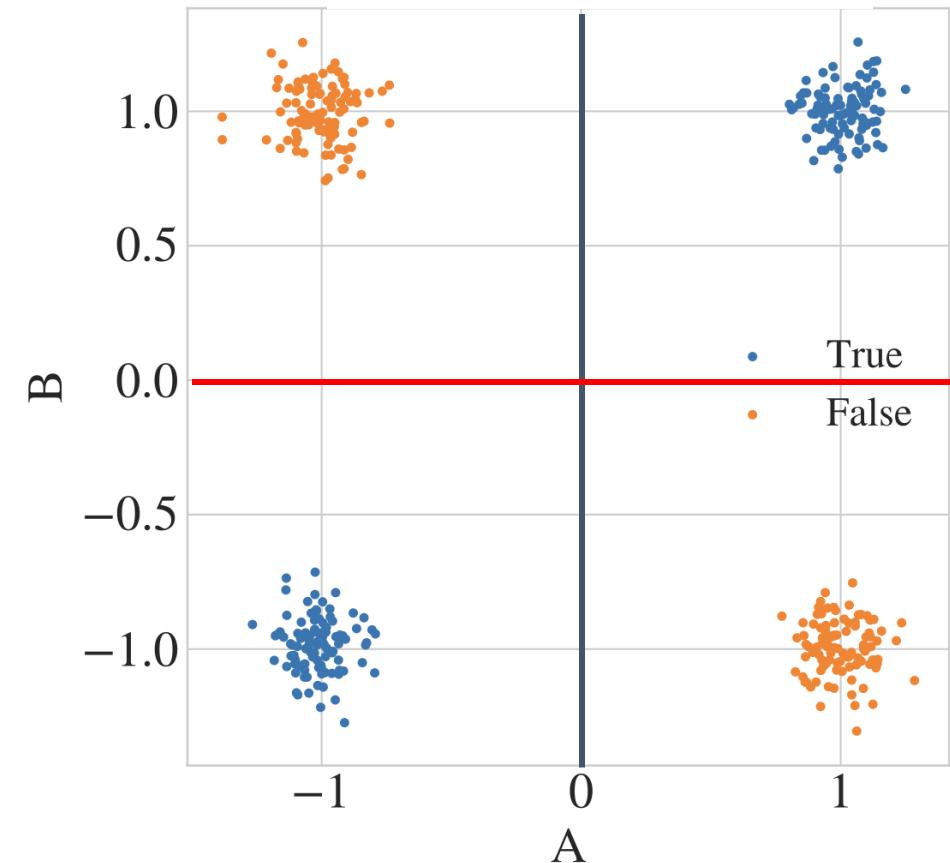
Decision trees

- Decision trees incrementally ask questions about the features to split the problem into smaller, simpler (binary) decisions



- All root and inner nodes question the value of a feature, and branches split the dataset into **different regions to which a datapoint can belong uniquely**

Noisy XNOR dataset



A more formal view of decision trees

47

Introduction to ML

Linear models

SL principles

Trees and neural networks

Risk optimisation

Decision trees

- More formally, at a given node in parent region R_t asking a question about the j^{th} feature, we create two regions:

$$R_1 = \{x \mid x_j < \alpha_t^j, x \in R_t\}$$

$$R_2 = \{x \mid x_j \geq \alpha_t^j, x \in R_t\}$$

- The parameters θ of decision trees are the threshold values at each nodes (the sequence of α)
- Decision tree minimise a criterion at each node of the tree: the **cross-entropy** (classification) or the **squared error** (regression)

Classification

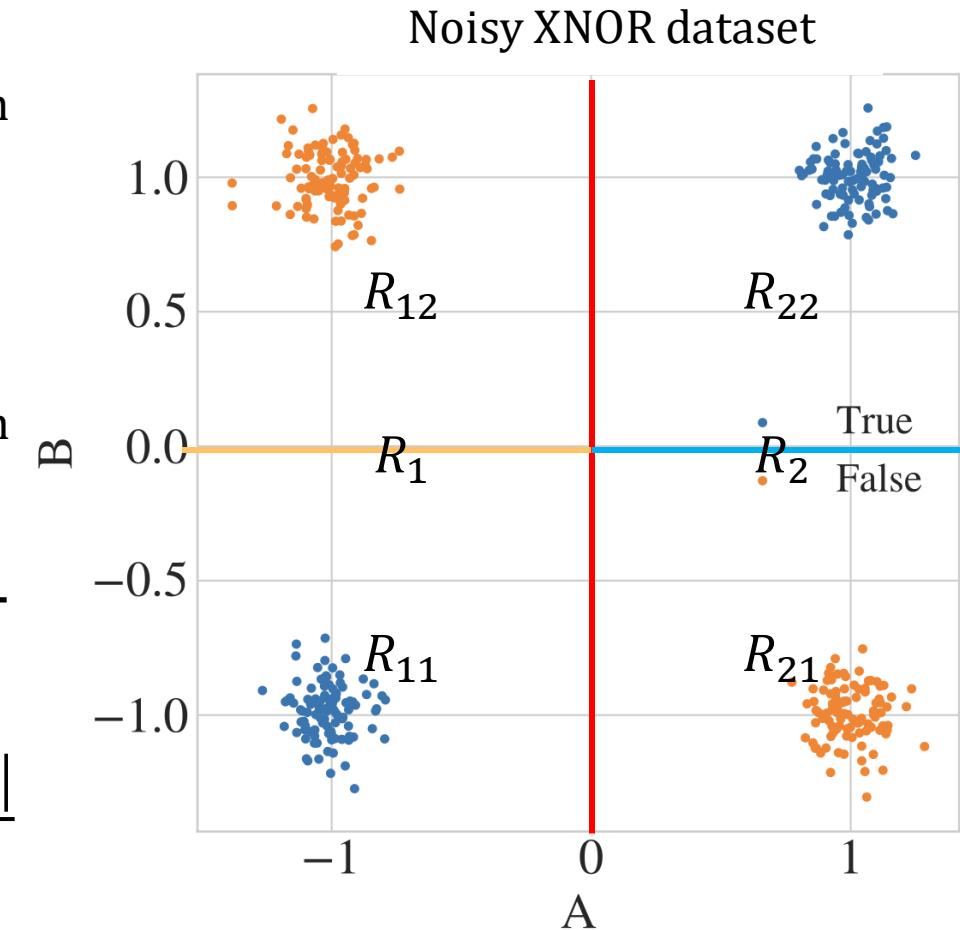
$$\ell(R_i) = - \sum_{k=1}^q \rho_k^i \log_2 \rho_k^i,$$

$$\rho_k^i = \frac{|\{x^{(j)} \mid x^{(j)} \in R_i, y^{(j)} = k\}|}{|\{x^{(j)} \mid x^{(j)} \in R_i\}|}$$

Regression

$$\ell(R_i) = \frac{1}{N} \sum_{j=1}^N (y_i - m)^2,$$

$$m = \frac{1}{N} \sum_{x^{(j)} \in R_i} y^{(j)}$$



A more formal view of decision trees

Introduction to ML

Linear models

SL principles

Trees and neural networks

Risk optimisation

Decision trees

- More formally, at a given node in parent region R_t asking a question about the j^{th} feature, we create two regions:

$$R_1 = \{x \mid x_j < \alpha_t^j, x \in R_t\}$$

$$R_2 = \{x \mid x_j \geq \alpha_t^j, x \in R_t\}$$

- The parameters θ of decision trees are the threshold values at each nodes (the sequence of α)
- Decision tree minimise a loss function at each node of the tree: the **cross-entropy** or the **squared error** (classification vs regression)

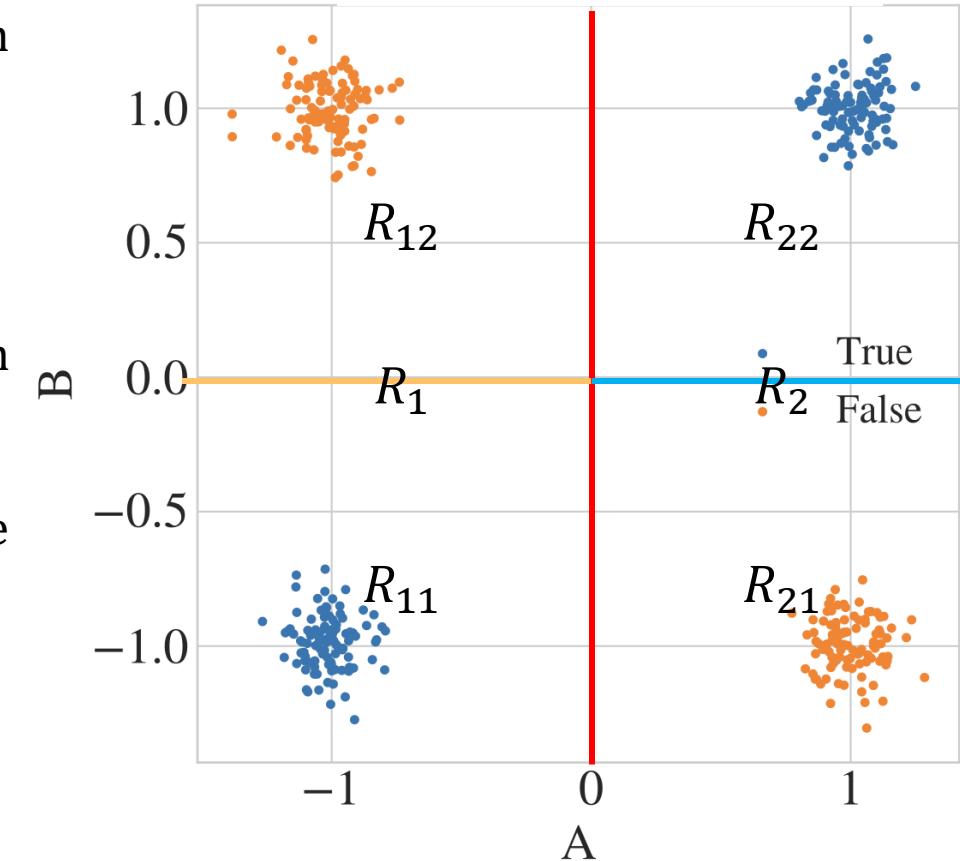


can handle categorical values, easy to interpret, fast to compute, both regression and classification



Shallow trees: high bias estimators (underfit), deep trees: high variance estimators (overfit)

Noisy XNOR dataset



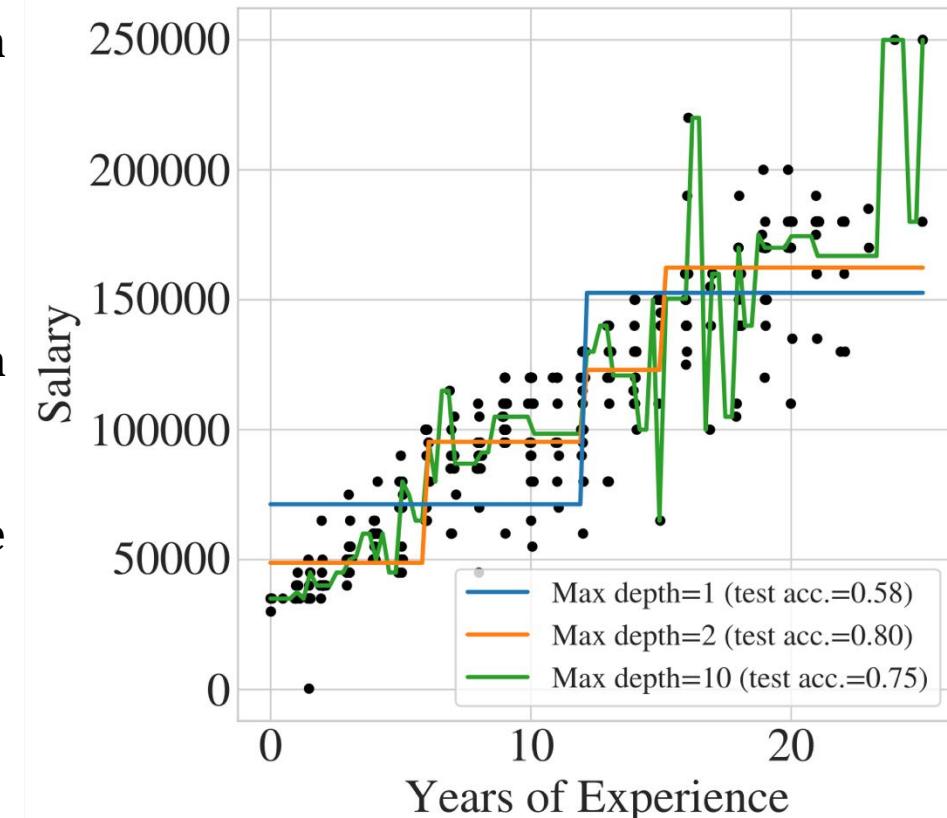
Decision trees

- More formally, at a given node in parent region R_t asking a question about the j^{th} feature, we create two regions:

$$R_1 = \{x \mid x_j < \alpha_t^j, x \in R_t\}$$

$$R_2 = \{x \mid x_j \geq \alpha_t^j, x \in R_t\}$$

- The parameters θ of decision trees are the threshold values at each nodes (the sequence of α)
- Decision tree minimise a loss function at each node of the tree: the **cross-entropy** or the **squared error** (classification vs regression)



can handle categorical values, easy to interpret, fast to compute, both **regression** and classification



Shallow trees: high bias estimators (underfit), deep trees: high variance estimators (overfit)

A more formal view of decision trees

50

Introduction to ML

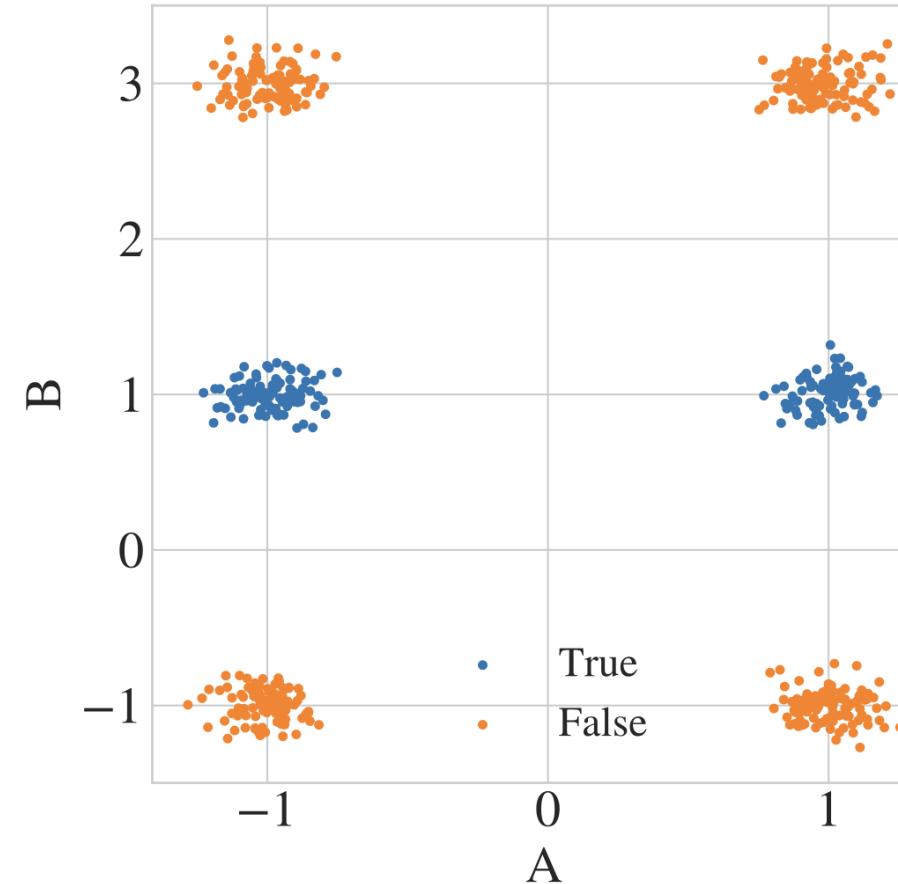
Linear models

SL principles

Trees and neural networks

Risk optimisation

Decision trees



Practice: Can you build a decision tree solving the binary classification problem?

Ensembling methods: bagging

51

Introduction to ML

Linear models

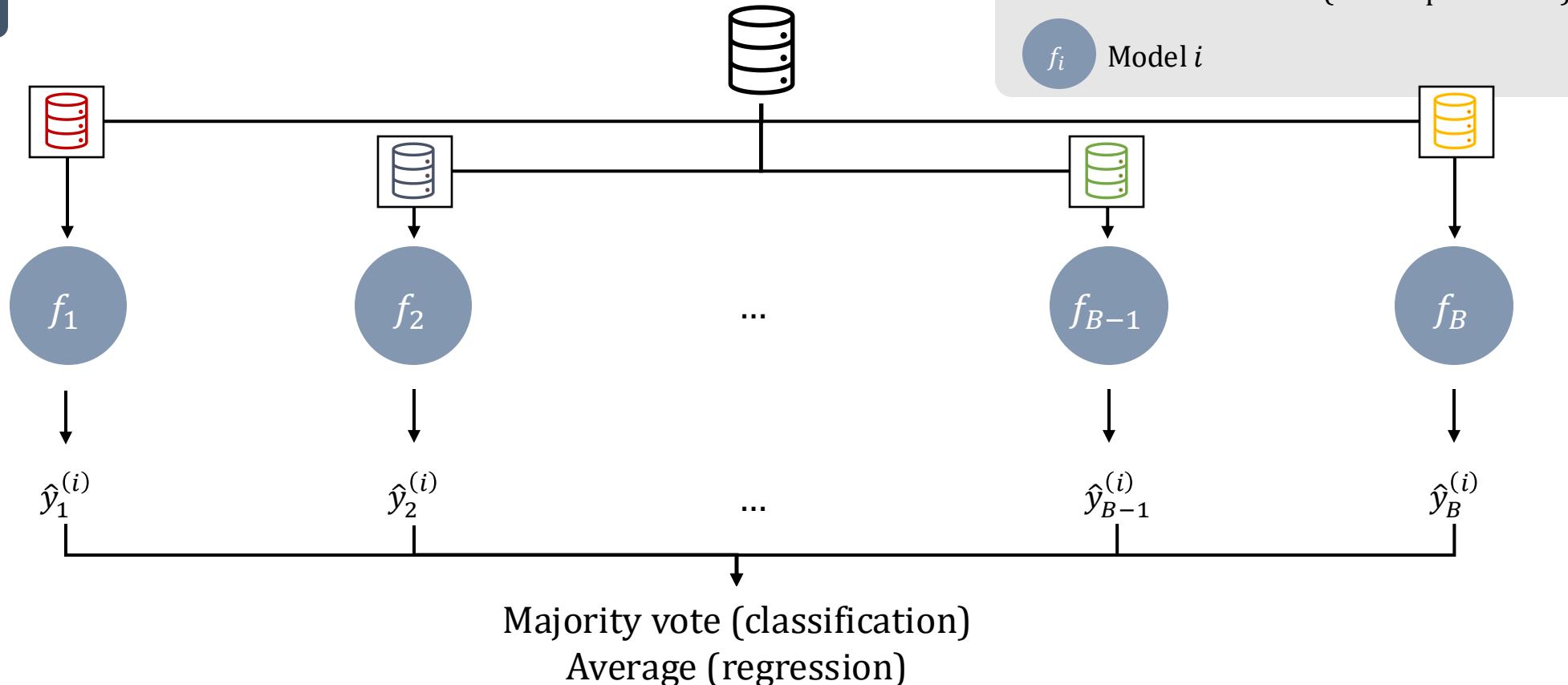
SL principles

Trees and neural networks

Risk optimisation

- To circumvent the problems that can have weak learners like decision trees, **ensembling methods** were proposed

Bagging



- To reduce the variance, models need to be **uncorrelated**: this is achieved by using **random sampling of the dataset**

Ensembling methods: random forest

52

Introduction to ML

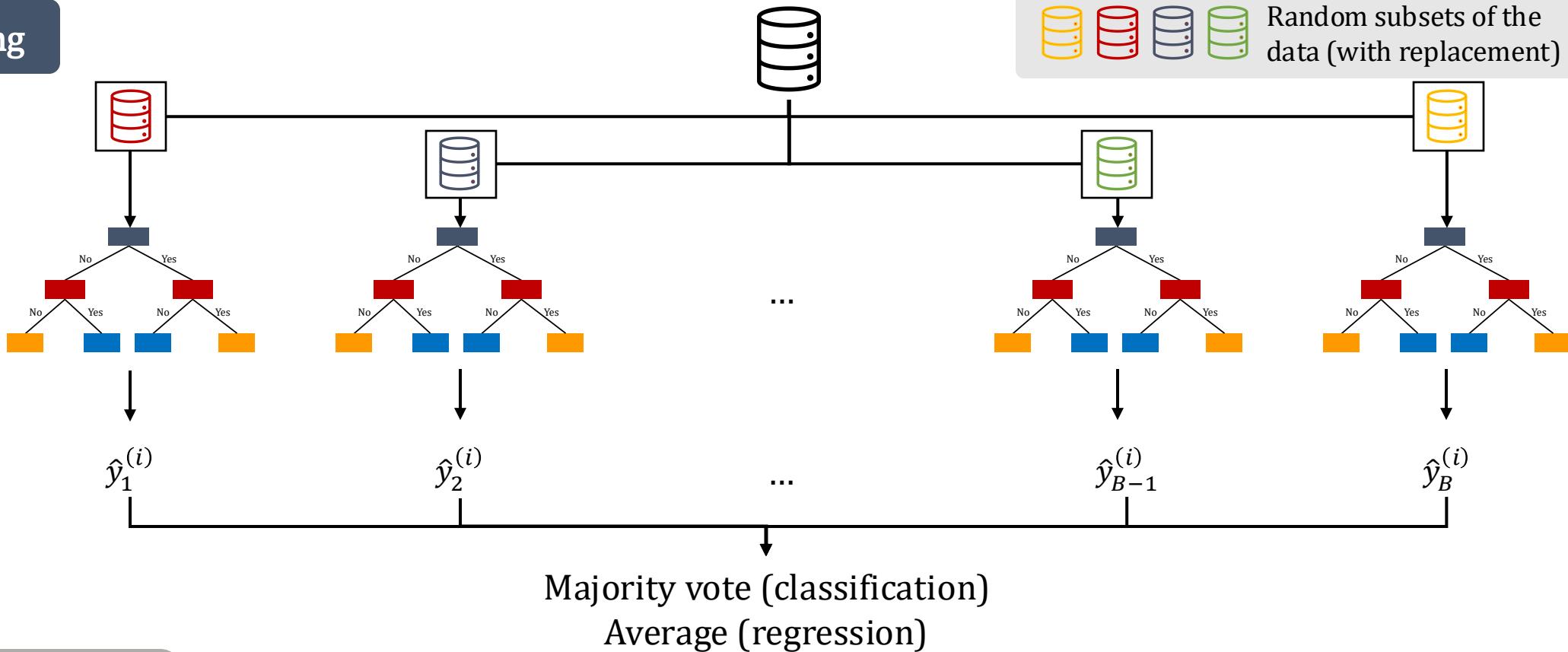
Linear models

SL principles

Trees and neural networks

Risk optimisation

Bagging



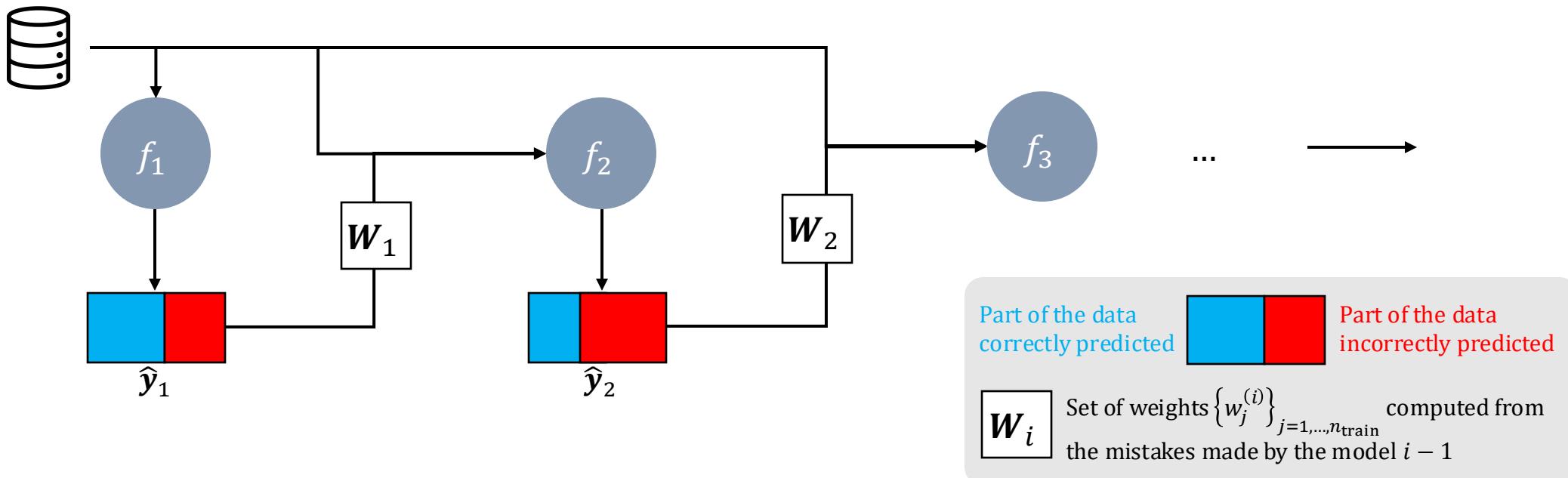
Random Forest

= Bagging of decision trees + feature bagging

- + Can still handle categorical values, both regression and classification, **reduced variance**
- More expensive to compute (need to train B trees instead of one), harder to interpret

- While bagging trains high-variance models in parallel to reduce the variance of the combined estimate, **boosting trains high-bias models sequentially to reduce the overall bias**

Boosting



- Successive learners f_i are fed by data X_i , a weighted version of the initial dataset X , **giving more weights to the errors committed by the previous model f_{i-1}**
- The output is, as in bagging, **a linear combination of all the learners** weighted by the contribution of each tree
- The choice of weighting and training depends on the algorithm and context (see [Adaboost](#) or [gradient boosting](#))

Ensembling methods: boosted trees

54

Introduction to ML

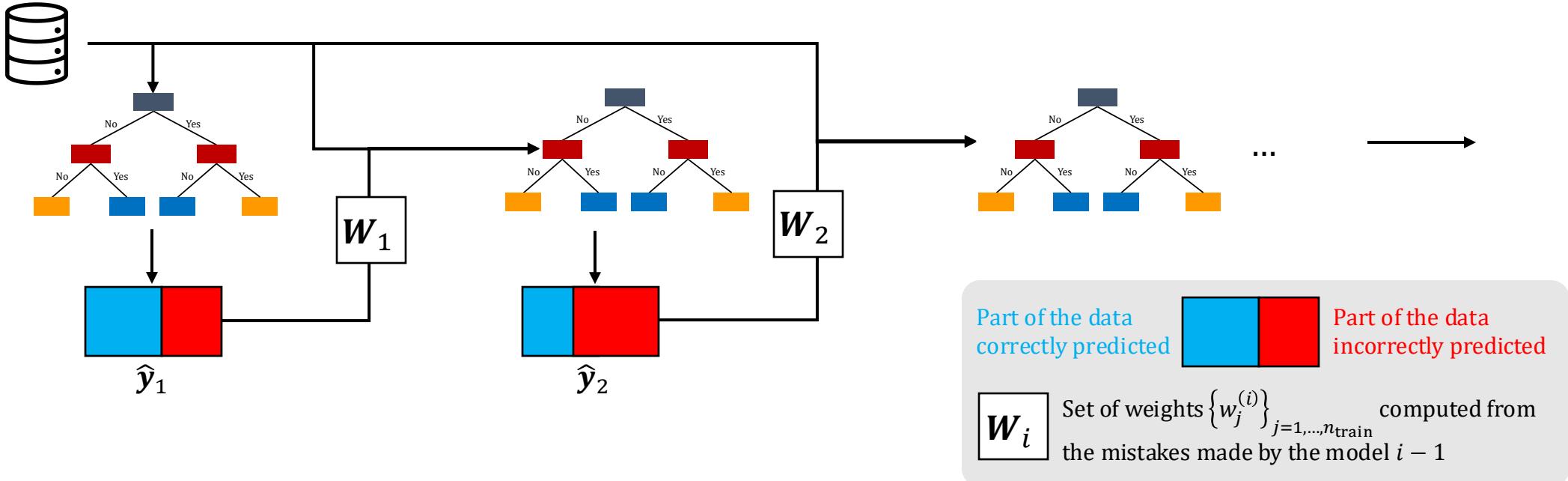
Linear models

SL principles

Trees and neural networks

Risk optimisation

Boosting

**Boosted trees**

= Boosting of decision trees

- + Both regression and classification (residuals or weighted classification error), **reduced bias**, good performances
- More expensive to compute, increased variance, subject to overfitting

Illustrative comparison

55

Introduction to ML

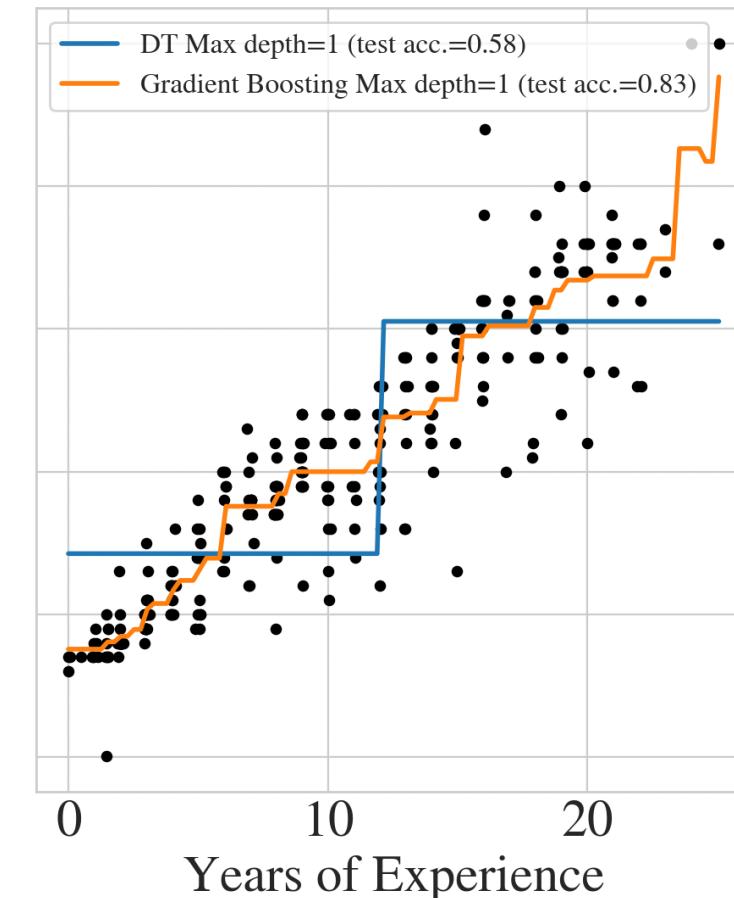
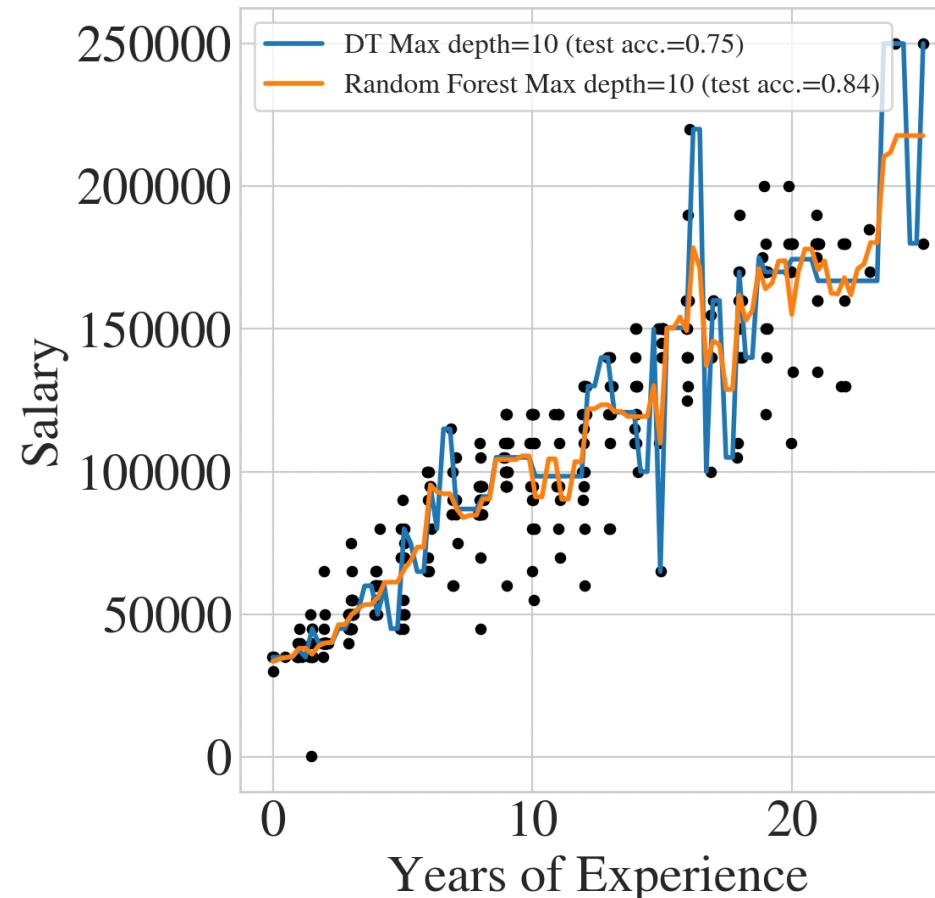
Linear models

SL principles

Trees and neural networks

Risk optimisation

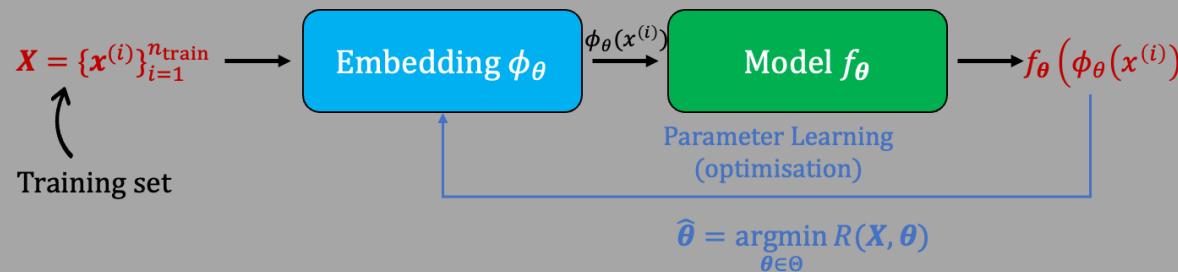
Comparison on our regression problem



Artificial neural networks

- How can we handcraft features $\phi(x)$ for complex problems like real-life image classification allowing the use of simple linear decision models?
- **The idea of neural network is to parameterise the embedding $\phi_\theta(x)$ and find the basis linearising the problem.**

Training phase



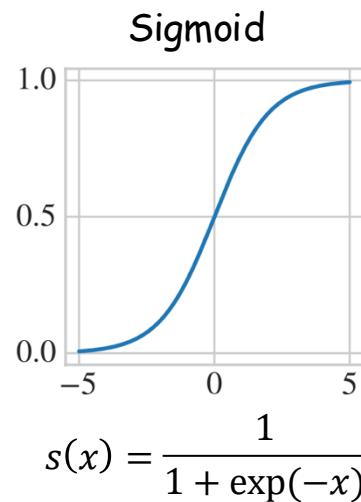
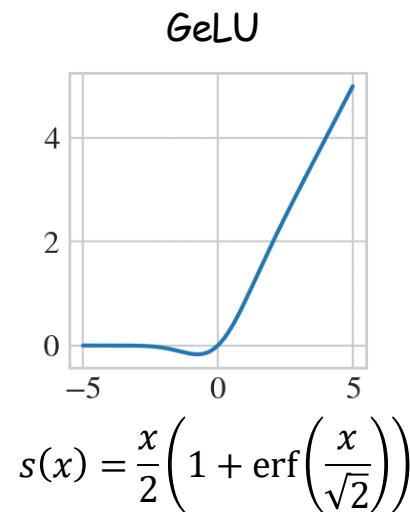
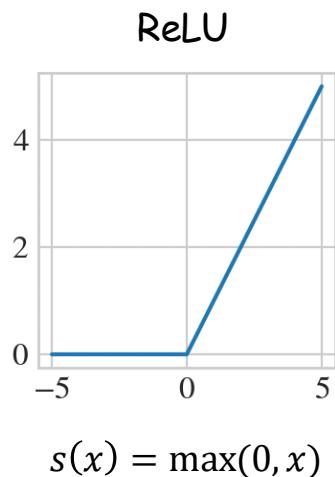
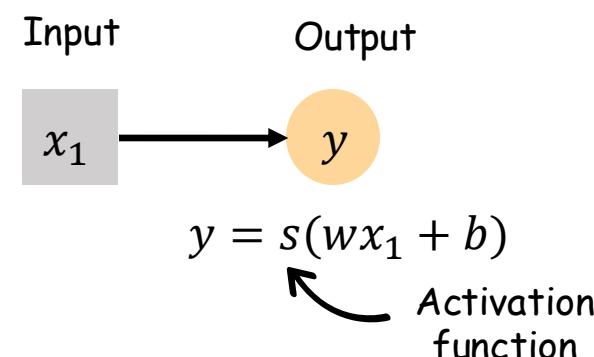
- Now our prior knowledge about the problem must go in the architecture of the network



Images from the ImageNet dataset

Artificial neural networks

- Building block of neural networks: the **neuron**
 - Made of three operations: it first multiplies the input by a **weight**, then adds a **bias**, and finally applies an **activation function** s to the result
 - If s is the identity, you recognize the linear regression model with $\theta_0 = b$ and $\theta_1 = w$. To represent non-linear functions, s must be non-linear



- This model is motivated by biological neurons and the activation function mimics the activation or inhibition through **non-linear (and differentiable) functions**
 - Can take different forms but some examples include the **ReLU or sigmoid functions**

Artificial neural networks: more features

58

Introduction to ML

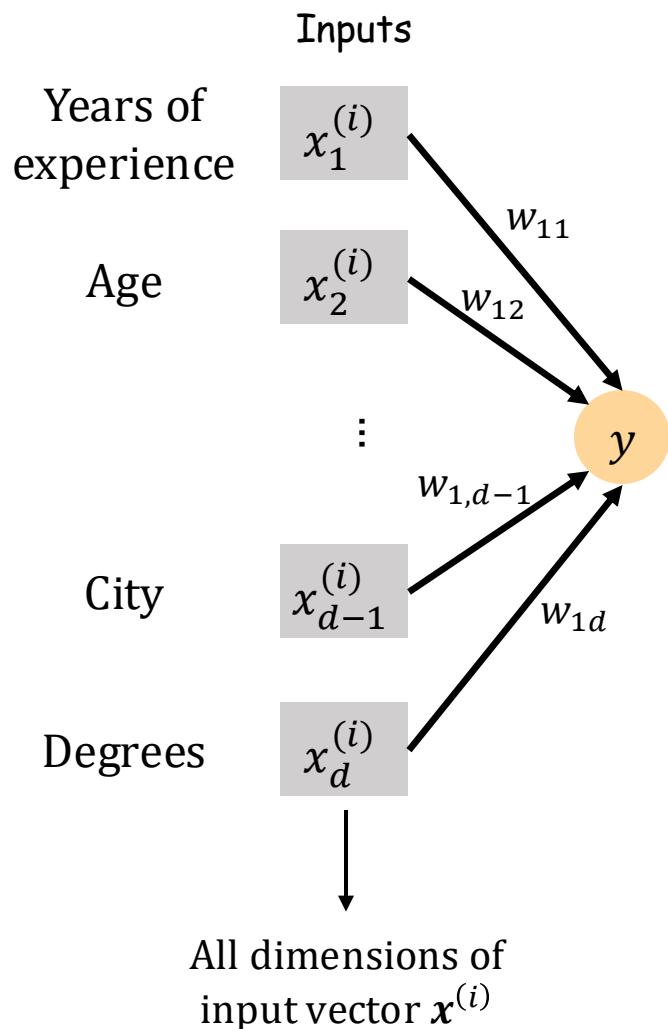
Linear models

SL principles

Trees and neural networks

Risk optimisation

Artificial neural networks



Notation: w_{ij} is the weight linking the unit j of the first layer to the unit i of the next layer

Output

$$y = s \left(\sum_{j=1}^d w_{1j} x_j^{(i)} + b \right)$$

$$y = s(\mathbf{w}_1 \cdot \mathbf{x}^{(i)} + b)$$

Such a network has $p = d + 1$ parameters
(as in multi-feature linear regression)

More layers: the art of learning features

59

Introduction to ML

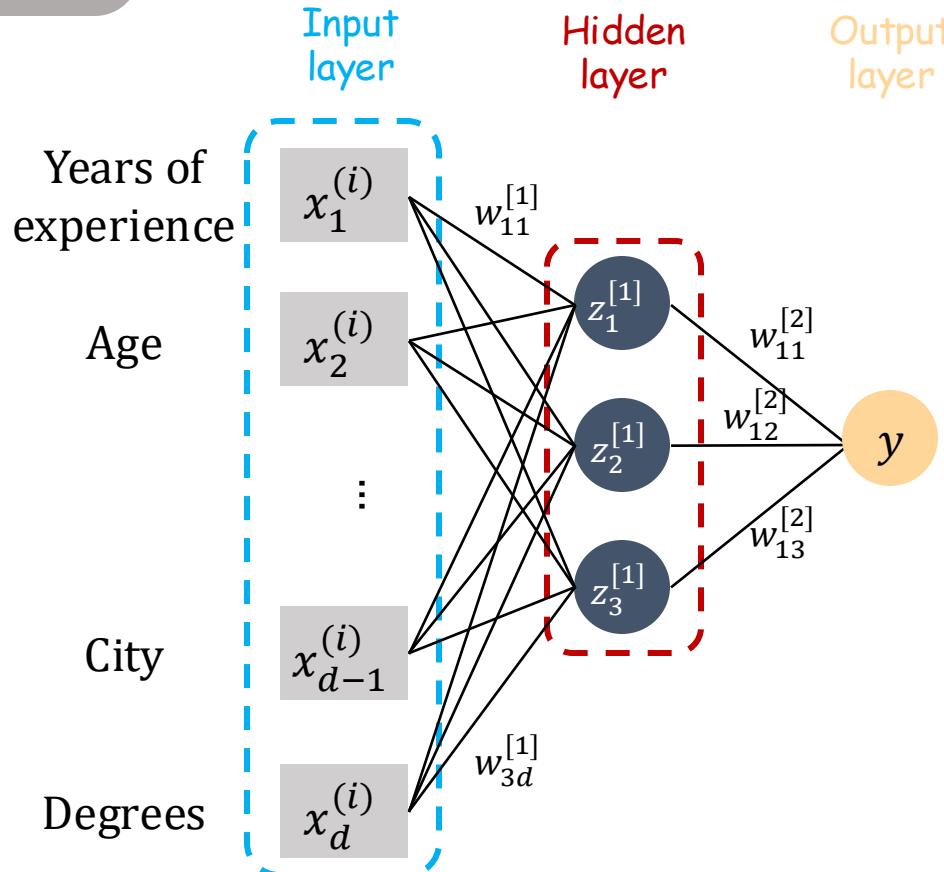
Linear models

SL principles

Trees and neural networks

Risk optimisation

Artificial neural networks



Years of experience
Age
City
Degrees



- Units in a same layer do not interact
- Data go from input to output in a **feed-forward way**
- The width of the output layer corresponds to the number of classes/values that you want to predict
- The **activation** of the unit j in layer one is given by

$$z_j^{[1]} = s(\mathbf{w}_j^{[1]} \cdot \mathbf{x}^{(i)} + b_j^{[1]})$$

Weights of layer 1 for unit j (vectorized)
Layer 1
Unit j
Bias of the unit j in layer 1

- We also define the **pre-activation**

$$u_j^{[1]} = \mathbf{w}_j^{[1]} \cdot \mathbf{x} + b_j^{[1]}$$

More layers: the art of learning features

60

Introduction to ML

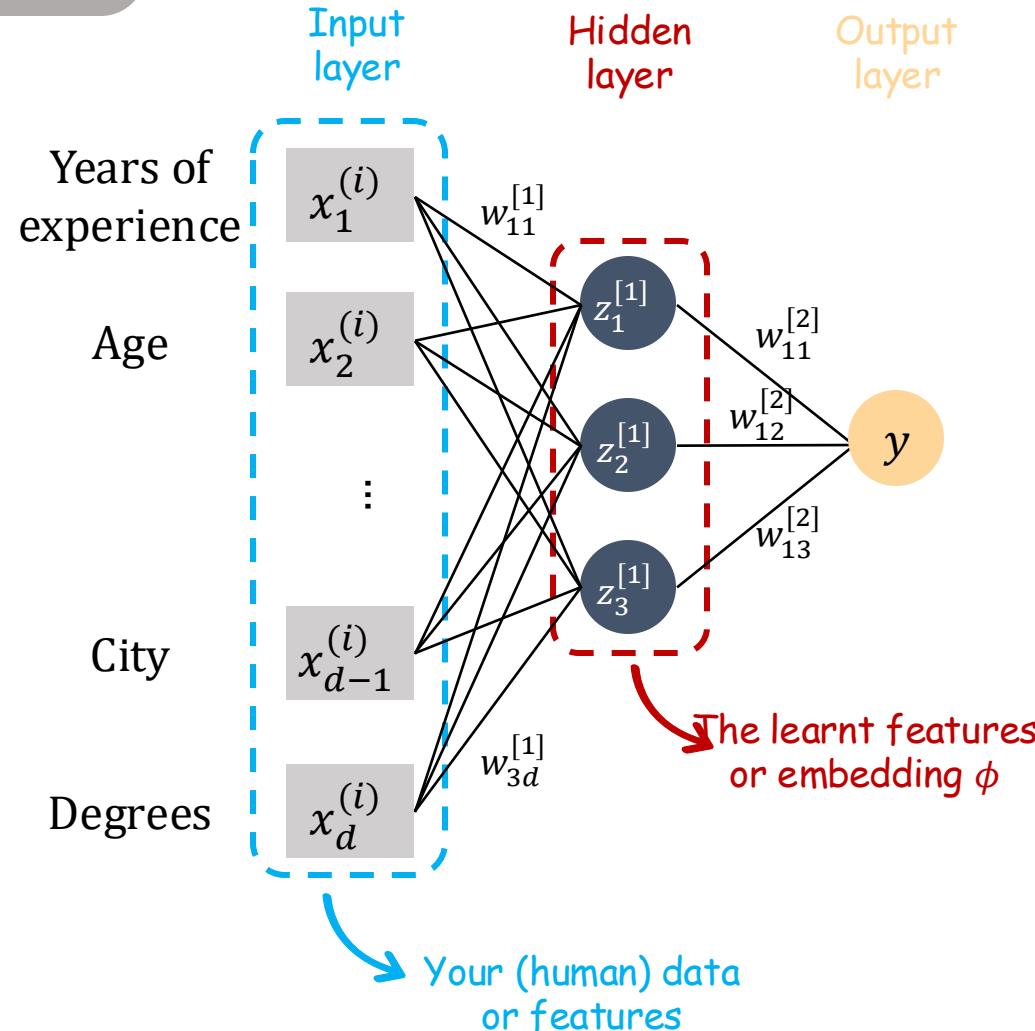
Linear models

SL principles

Trees and neural networks

Risk optimisation

Artificial neural networks



Let's have a look at the output of this network

$$y = s \left(\sum_{j=1}^d w_{1j}^{[2]} z_j^{[1]} + b^{[2]} \right)$$

$\{z_j\}_{j=1, \dots, 3}$ act as **new features** to predict y

General view of deep neural networks

Introduction to ML

Linear models

SL principles

Trees and neural networks

Risk optimisation

Artificial neural networks

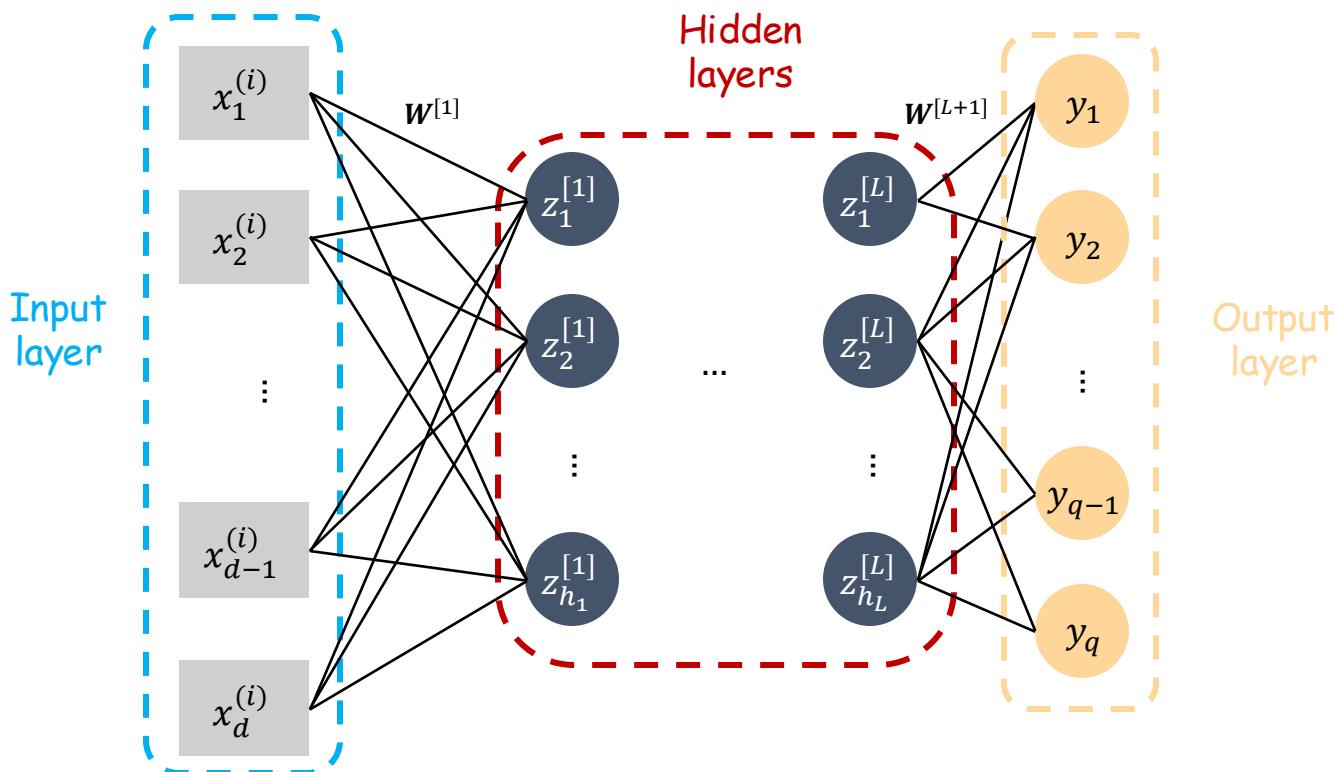
- A fully-connected neural network with d inputs, L hidden layers of width h_1, h_2, \dots, h_L and an output layer of size q
- The j^{th} output is computed as

$$\mathbf{y} = s^{[L+1]}(\mathbf{W}^{[L+1]}\mathbf{z}^{[L]}) \quad \text{with } \mathbf{z}^{[L]} = [1, z_1^{[L]}, z_2^{[L]}, \dots, z_{h_L}^{[L]}]^T$$

$$\mathbf{y} = s^{[L+1]} \left(\mathbf{W}^{[L+1]} s^{[L]} \left(\mathbf{W}^{[L]} \dots s^{[1]}(\mathbf{W}^{[1]}\mathbf{x}) \right) \right)$$

$$\mathbf{w}_j^{[l]} = [b_j^{[l]}, w_{1j}^{[l]}, w_{2j}^{[l]}, \dots, w_{h_L j}^{[l]}]$$

$$\mathbf{W}^{[l]} = [\mathbf{w}_1^{[l]}, \mathbf{w}_2^{[l]}, \dots, \mathbf{w}_{h_l}^{[l]}]^T \in \mathbb{R}^{h_l \times (h_{l-1} + 1)}$$



- In the end, a **neural network** is a, as other models, a function $f_{\theta}: X \rightarrow Y$ of some parameters ($\theta = \text{weights and biases}$)
- f_{θ} is a composition of non-linear function when s is non-linear, allowing to build non-linear estimators
- The parameters of the model are obtained by minimising the empirical risk (**cross-entropy for classification, MSE for regression**)

General view of deep neural networks

62

Introduction to ML

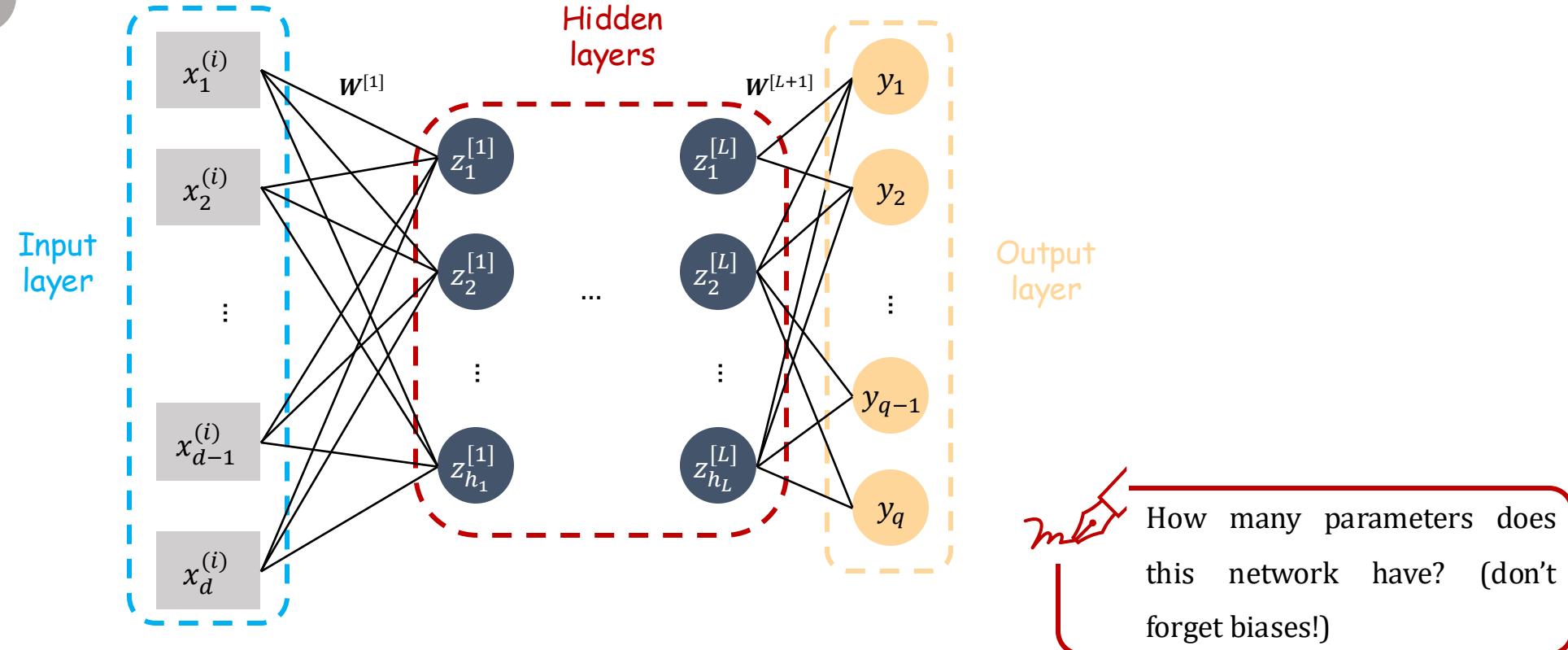
Linear models

SL principles

Trees and neural networks

Risk optimisation

Artificial neural networks



+ Both regression and classification, learns features, good performances when enough data

- Need a lot of data to train, subject to overfitting, uninterpretable, targeted-purpose architectures usually work better

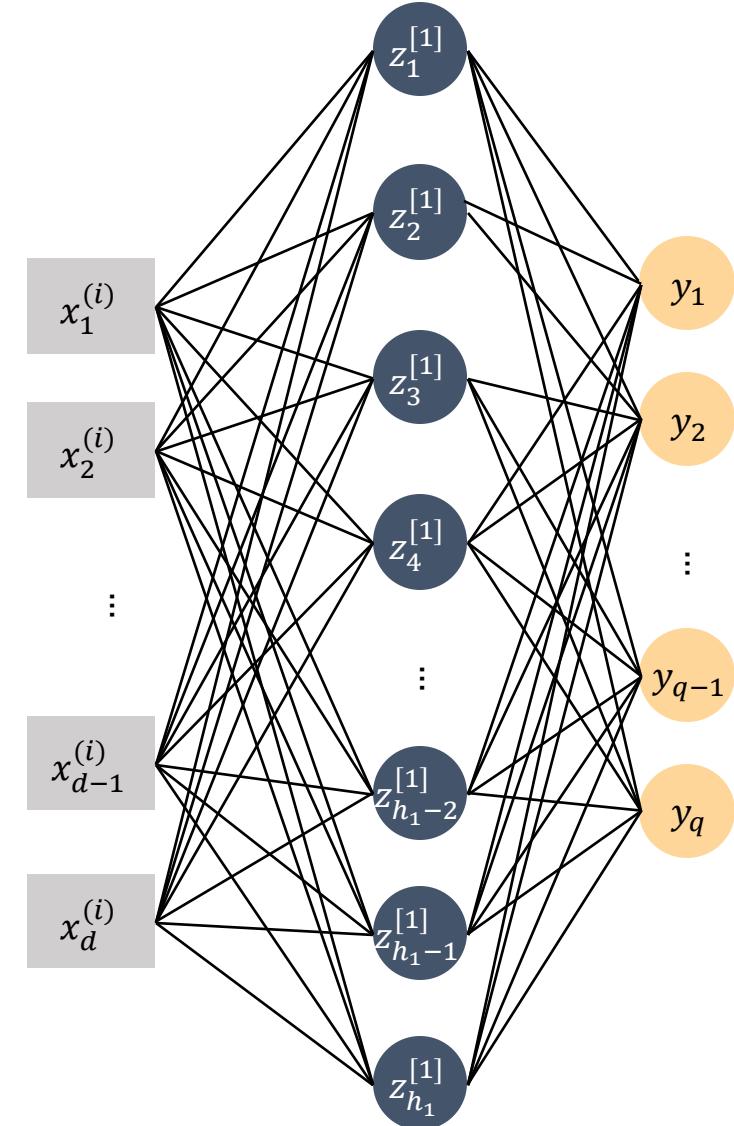
Artificial neural networks

- Single-layer neural networks are universal approximation (see [Cybenko 1989](#))

Theorem (informal)

*A fully-connected neural network with a single hidden layer ($L = 1$) with enough neurons (h_1 large) can fit **any arbitrary smooth function.***

- In practice, this does **not** help: the width may need to scale exponentially with the function complexity.
- Empirically, deep networks are more efficient than wide ones.



ANN: example on the XNOR dataset

64

Introduction to ML

Linear models

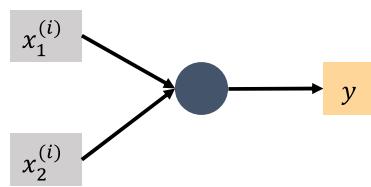
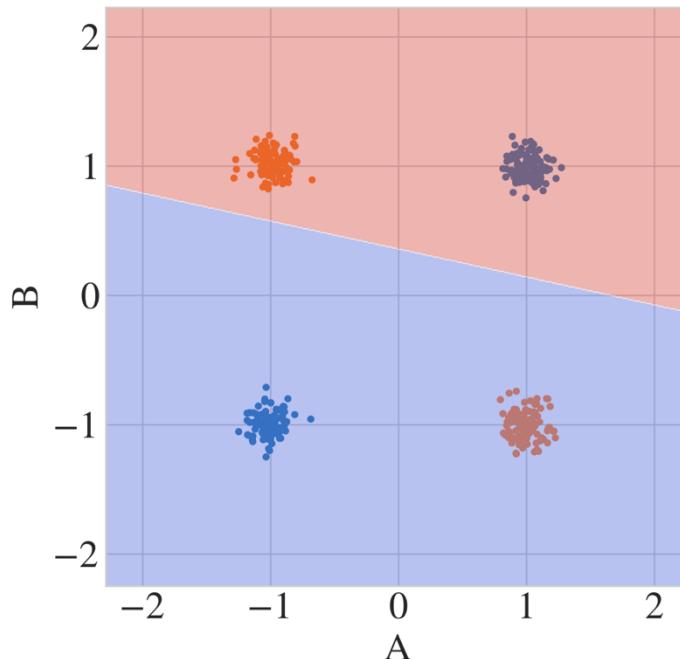
SL principles

Trees and neural networks

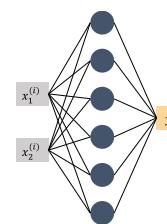
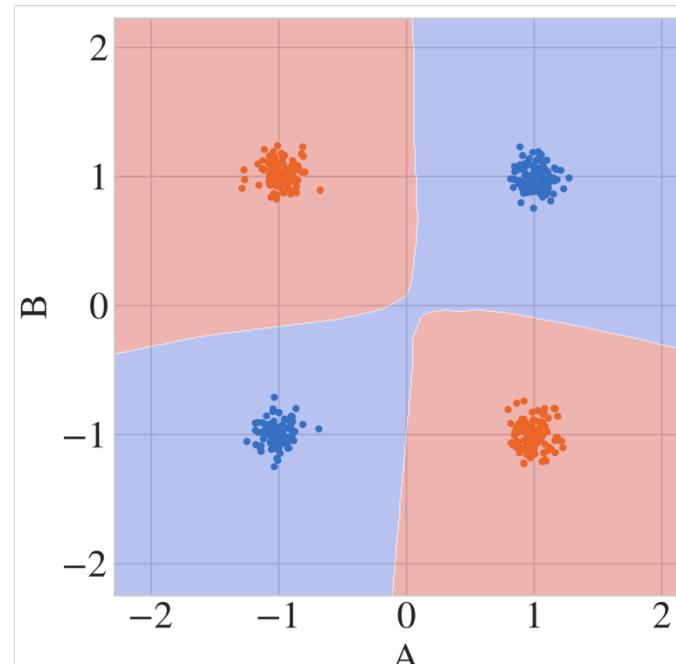
Risk optimisation

Artificial neural networks

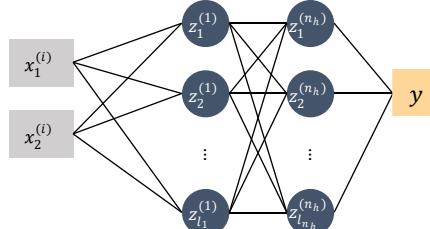
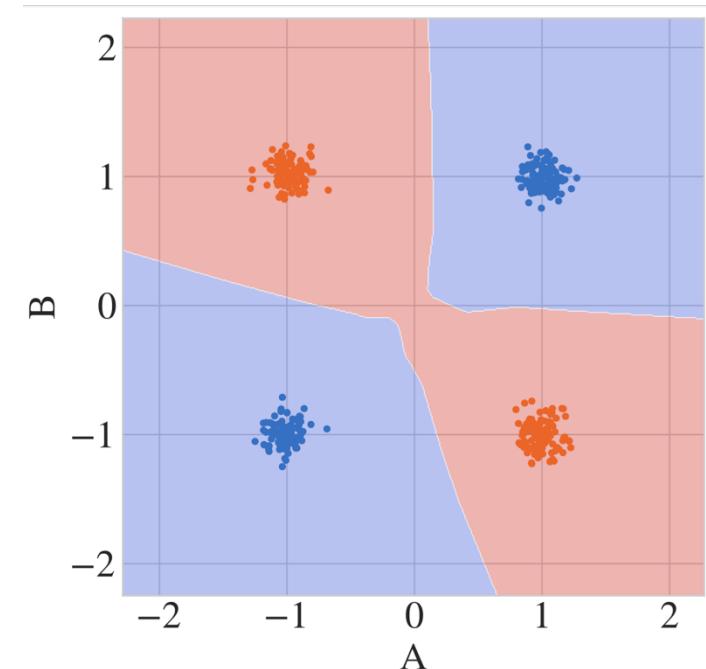
1 hidden layer of 1 unit



1 hidden layer of 100 units



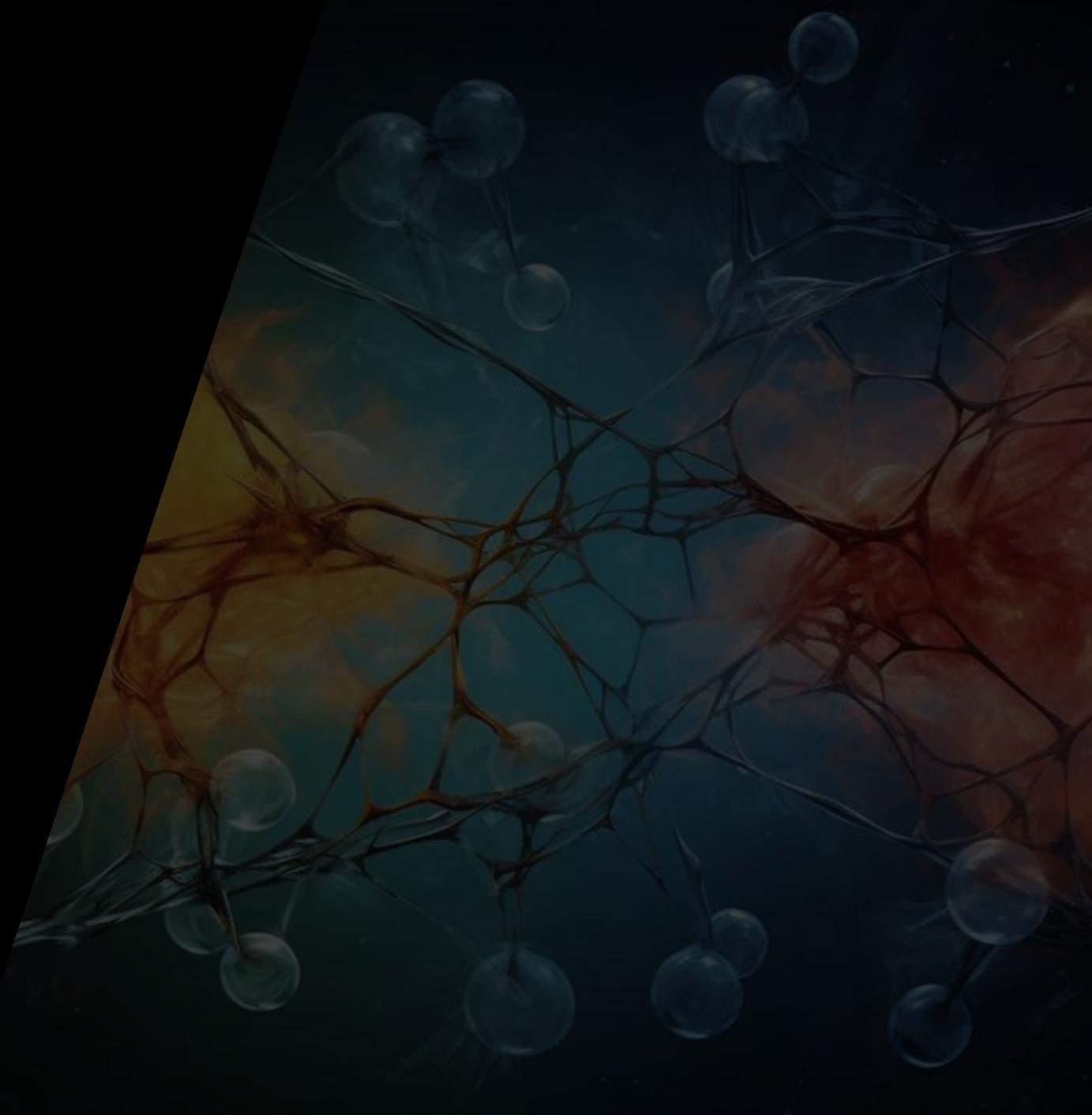
2 hidden layers of 10 units



Optimisation with ERM

Contents:

- *Empirical risk minimization for parameter learning*
- *Gradient descent*
- *Stochastic gradient descent*
- *Backpropagation for computing gradients in neural networks*



How to perform ERM in general?

Introduction to ML

Linear models

SL principles

Trees and neural networks

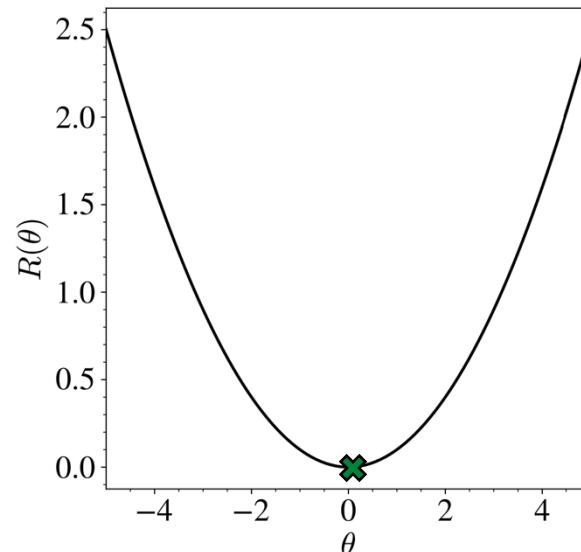
Risk optimisation

- The aim of learning in supervised learning is to compute

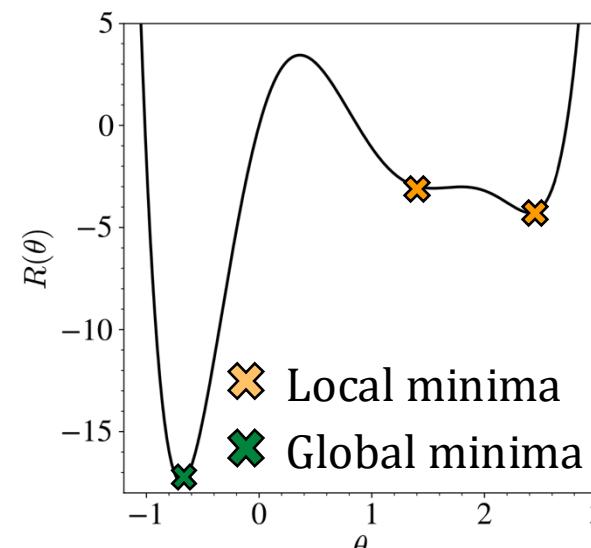
$$\hat{\boldsymbol{\theta}} = \operatorname{argmin}_{\boldsymbol{\theta}} R(\boldsymbol{\theta}) = \operatorname{argmin}_{\boldsymbol{\theta}} \frac{1}{n_{\text{train}}} \sum_{i=1}^{n_{\text{train}}} \ell(f_{\boldsymbol{\theta}}(\boldsymbol{\phi}^{(i)}), y^{(i)}).$$

- In linear regression, we could directly optimise the empirical risk in closed-form, but **in general it is not possible**.
- How to perform the **Empirical Risk Minimisation** (ERM) in general?

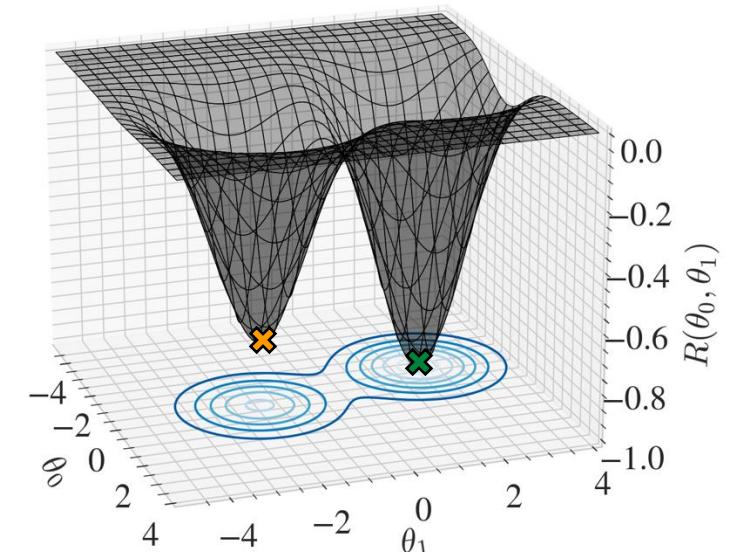
1D convex



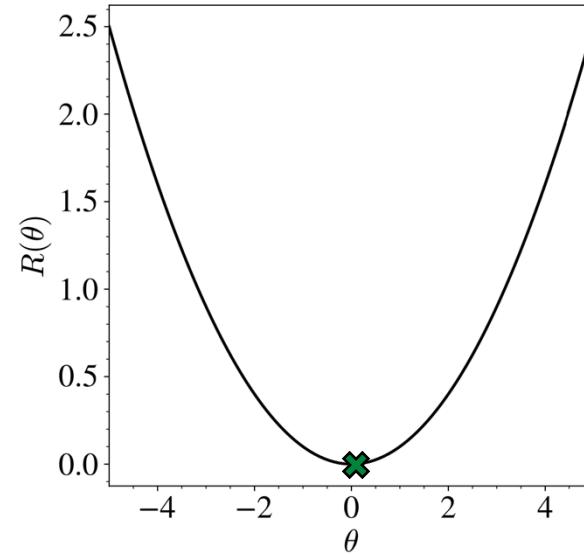
1D non-convex



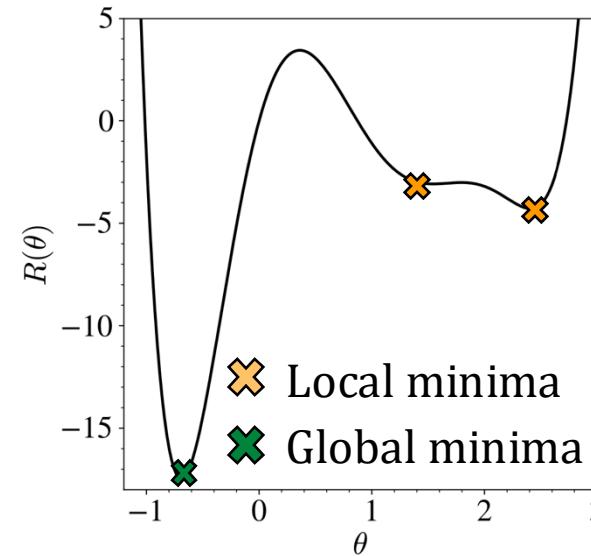
2D non-convex



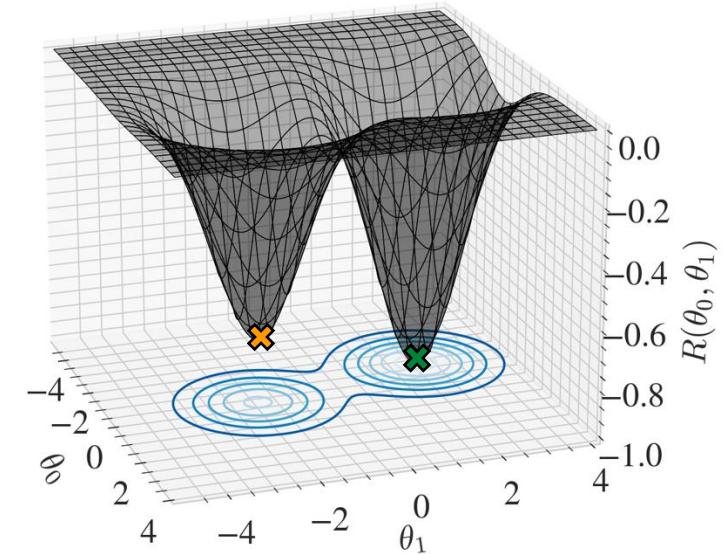
1D convex



1D non-convex



2D non-convex



- A naïve way of minimizing such functions could be to uniformly pave the parameter space and choose the one with the smallest value as being the minimum: this is **global optimization (grid search optimization)**
- This is where the **curse of dimensionality** kicks in. In general, we optimize models over many parameters $d \gg 1$ (imagine the pixels of an image) and all the points are far away from each other in high dimensions
- Sampling uniformly $[0, 1]^{10}$ with a step of 0.01 requires 10^{20} evaluations (think of GPT-3 and its 175 billion parameters!)

Gradient descent algorithm

Introduction to ML

Linear models

SL principles

Trees and neural networks

Risk optimisation

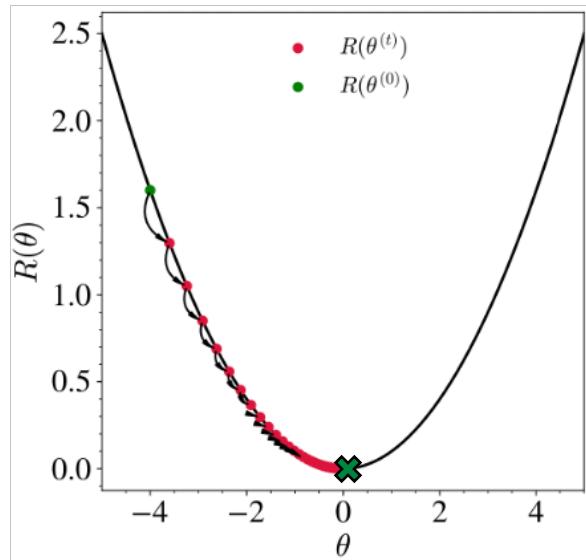
- A solution: local, directed search to navigate in the landscape
 - Numerical optimisation by **gradient descent**

Note: the superscript does not have to do with the training example here but with the time step (θ is a parameter).

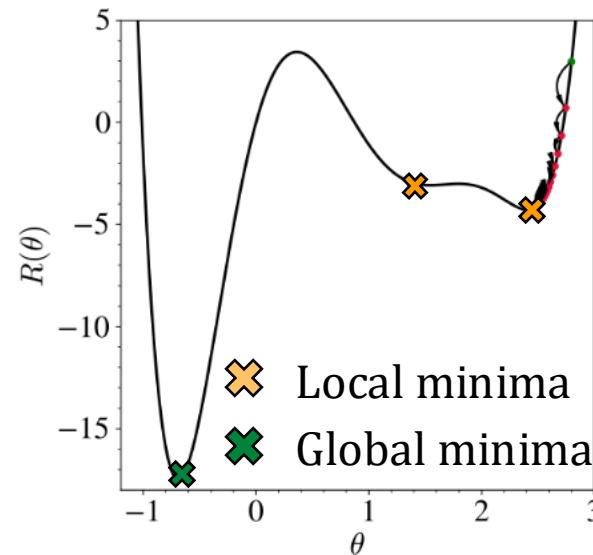
Algorithm: Gradient descent

- Initialise θ_0 randomly
- Compute $\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_{\theta} R(\theta^{(t)})$
- Repeat 2 until a stopping criterion is satisfied

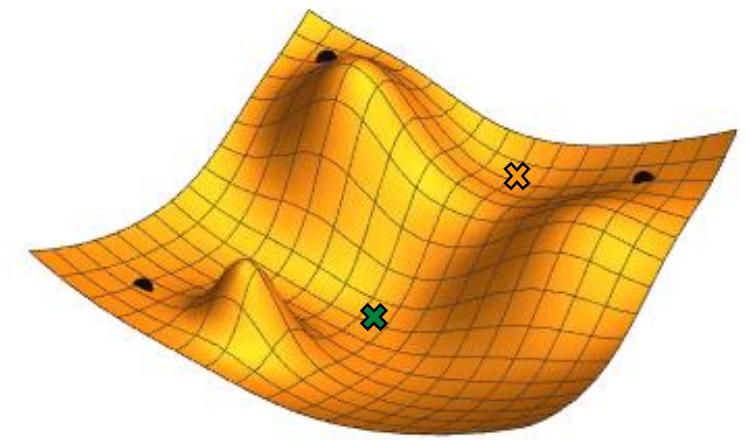
1D convex



1D non-convex



2D non-convex



Solutions in complex non-convex models depend on θ_0 : it is an **hyperparameter**

- Example of stopping criterion: $\|\theta^{(t+1)} - \theta^{(t)}\|_2^2 \leq \epsilon$, or fixed number of steps

A word about hyperparameters

Introduction to ML

Linear models

SL principles

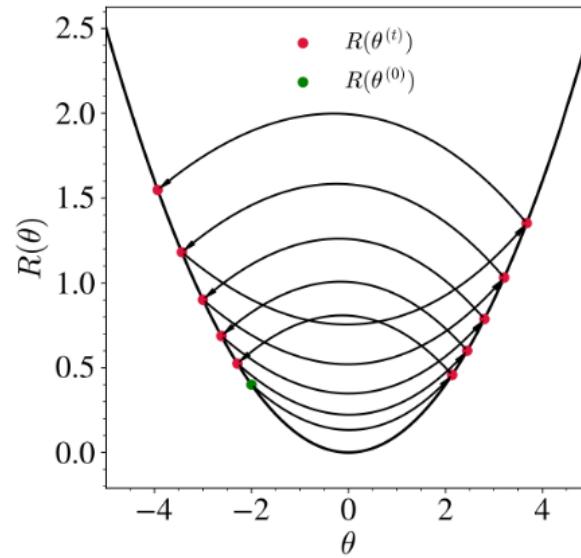
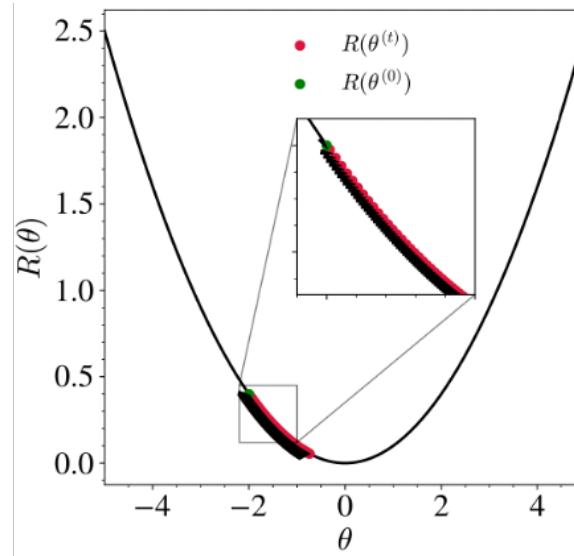
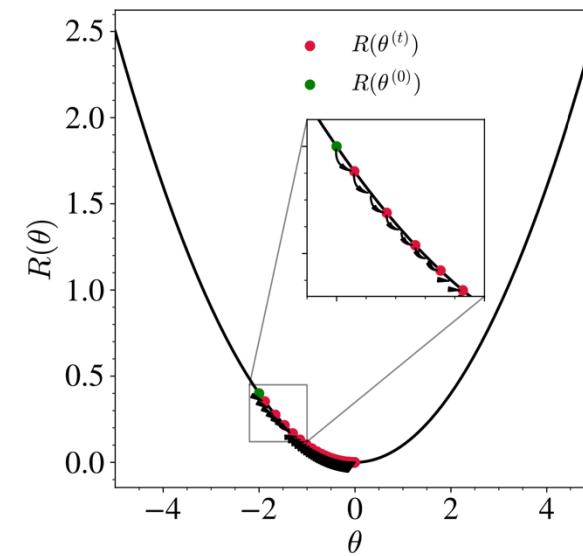
Trees and neural networks

Risk optimisation

- One **hyperparameter**: the learning rate η
- A value that is too large can lead to divergence while, when too small, the computational cost explodes (+ stuck in small asperities)



Hyperparameter: parameter that is **not learned** during the optimisation.
Ex: depth of tree in DTs, # of trees in RFs, learning rates, etc.

 η too large η too smallappropriate η 

- Usually, we use **grid search** to find the hyperparameter performing best on a third dataset: the **validation set (or use cross-validation)**

Example: our linear regression

70

Introduction to ML

Linear models

SL principles

Trees and neural networks

Risk optimisation

- In linear regression, the empirical risk is

$$R(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^n (\theta_0 + \theta_1 x_1^{(i)} - y^{(i)})^2 \quad \boldsymbol{\theta} = [\theta_0, \theta_1]^T$$

- Let's apply the gradient descent algorithm starting from $\boldsymbol{\theta}^{(0)}$ random
- Then, we need to compute the gradient $\nabla_{\boldsymbol{\theta}} R(\boldsymbol{\theta})$

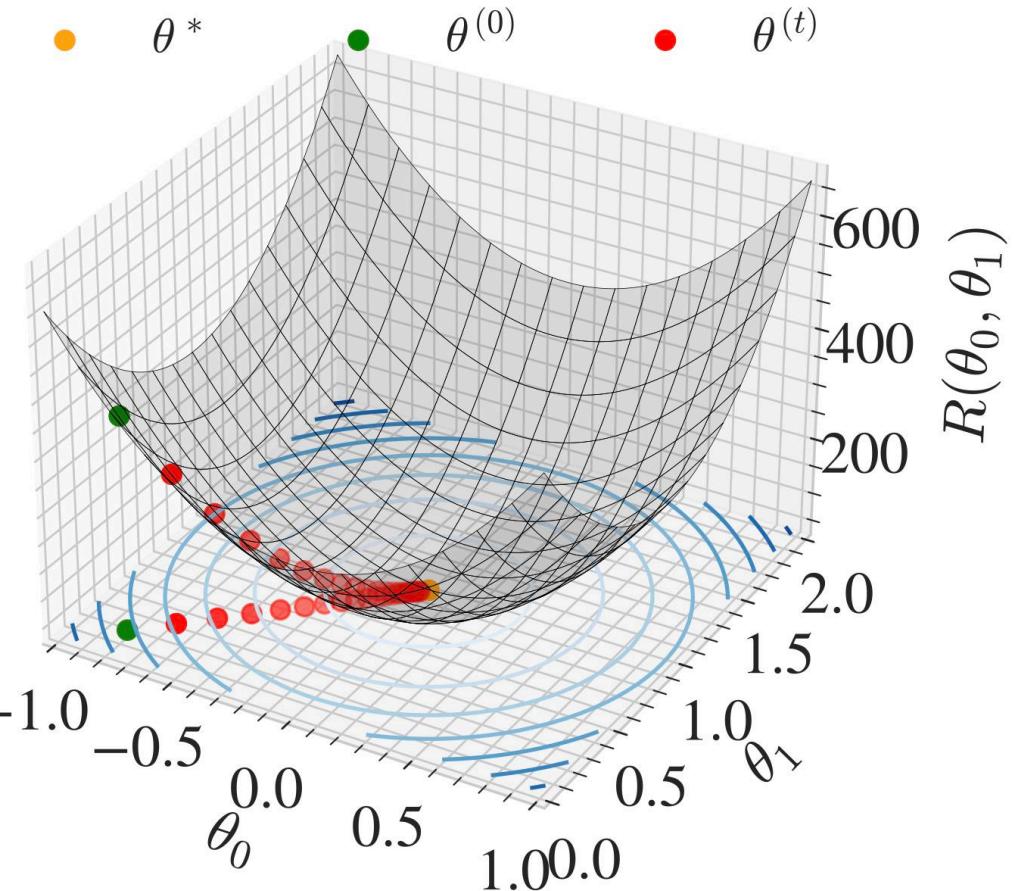
$$\frac{\partial R(\boldsymbol{\theta})}{\partial \theta_0} = \frac{2}{n} \sum_{i=1}^n (\theta_0 + \theta_1 x_1^{(i)} - y^{(i)}),$$

$$\frac{\partial R(\boldsymbol{\theta})}{\partial \theta_1} = \frac{2}{n} \sum_{i=1}^n x_1^{(i)} (\theta_0 + \theta_1 x_1^{(i)} - y^{(i)})$$

- Therefore, the update is

$$\theta_j^{(t+1)} = \theta_j^{(t)} - \frac{2\eta}{n} \sum_{i=1}^n x_j^{(i)} (\theta_0 + \theta_1 x_1^{(i)} - y^{(i)})$$

where $\forall i \in [1, \dots, n]$ $x_0^{(i)} = 1$



Note that I **standardized** the features. This is sometimes required when optimising some models. In GD, it allows faster convergence.

Stochastic gradient descent algorithm

71

Introduction to ML

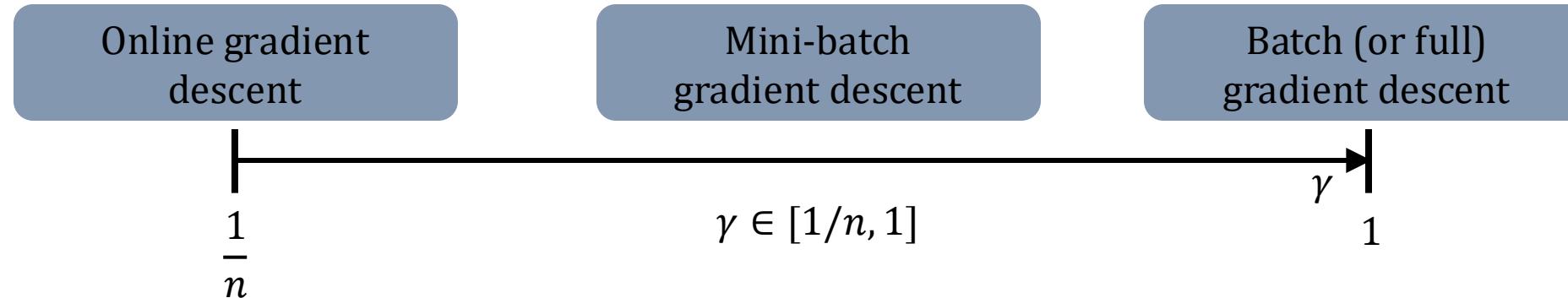
Linear models

SL principles

Trees and neural networks

Risk optimisation

- Problem of gradient descent: we **need the entire dataset** to compute $\nabla_{\theta} R(\theta^{(t)})$
- Solution: what about using only a fraction γ of the dataset chosen randomly?



Algorithm: Stochastic gradient descent

1. Initialise θ_0 randomly
2. For $e \in [1, \dots, E]$
 - 2.1. Shuffle the dataset
 - 2.2. For $i \in [1, \dots, |\gamma n|]$
 - 2.2.1. Compute $\theta^{(i+1)} = \theta^{(i)} - \eta \widehat{\nabla}_{\theta} R(\theta^{(i)})$
where $\widehat{\nabla} R(\theta) = \sum_{j=i\gamma n}^{i\gamma n + \gamma n} \nabla R_j(\theta)$

- e is called an **epoch** and a set of γn training examples is called a **mini-batch**
- Usually, **SGD often converges faster than full-batch GD**
- It may however oscillate around a true minimum
- Under some technical assumptions, **SGD provides an almost sure convergence** to a local (resp. global) in non-convex (resp. convex) landscapes

SGD landscapes on our linear regression

72

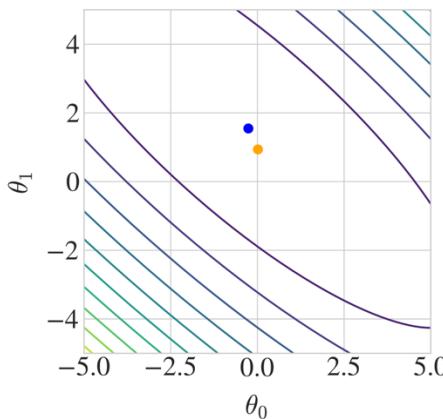
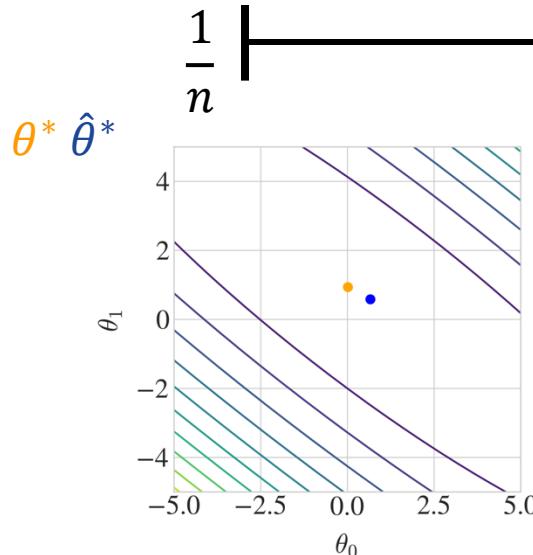
Introduction to ML

Linear models

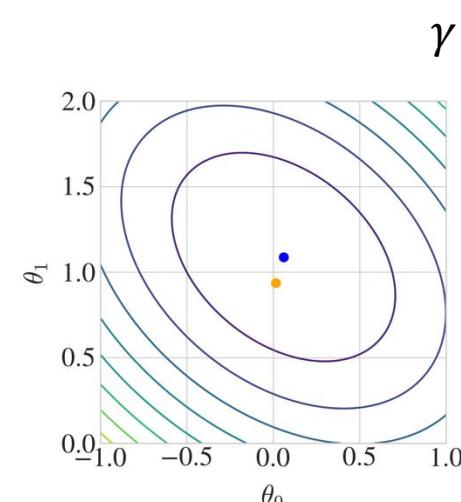
SL principles

Trees and neural networks

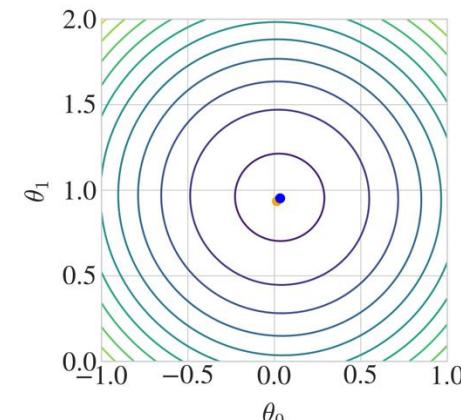
Risk optimisation

Online gradient
descentMini-batch
gradient descentBatch (or full)
gradient descent

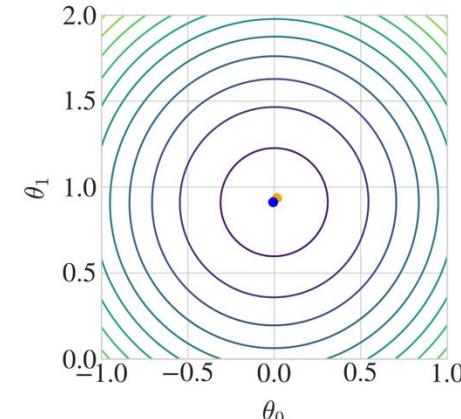
$$\gamma = 2/n$$



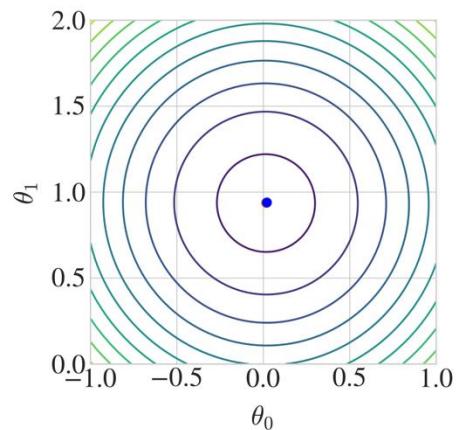
$$\gamma = 1/n$$



$$\gamma = 0.5$$



$$\gamma = 1$$



SGD on our linear regression

73

Introduction to ML

Linear models

SL principles

Trees and neural networks

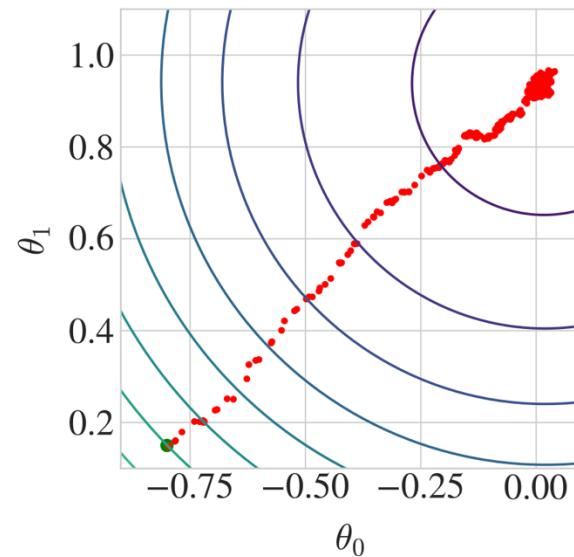
Risk optimisation

Online gradient
descentMini-batch
gradient descentBatch (or full)
gradient descent

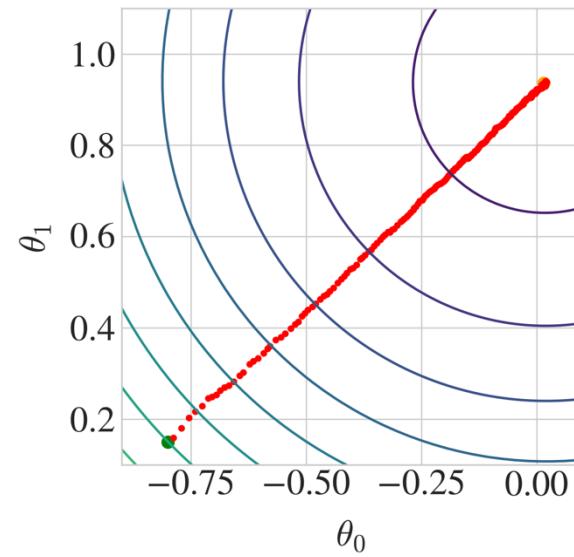
$$\frac{1}{n}$$

$$\gamma \in [1/n, 1]$$

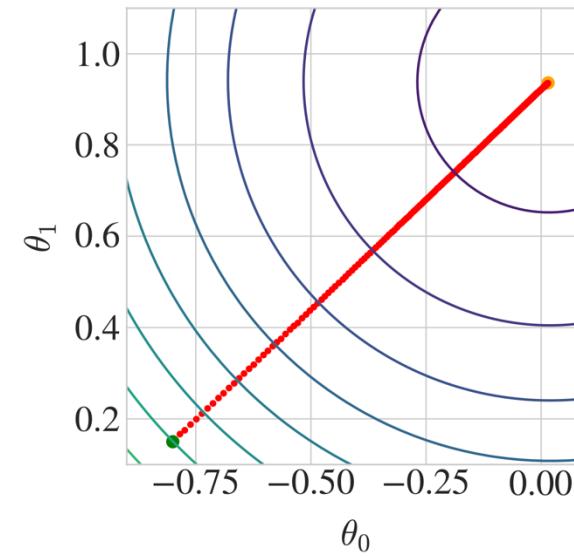
$$1$$



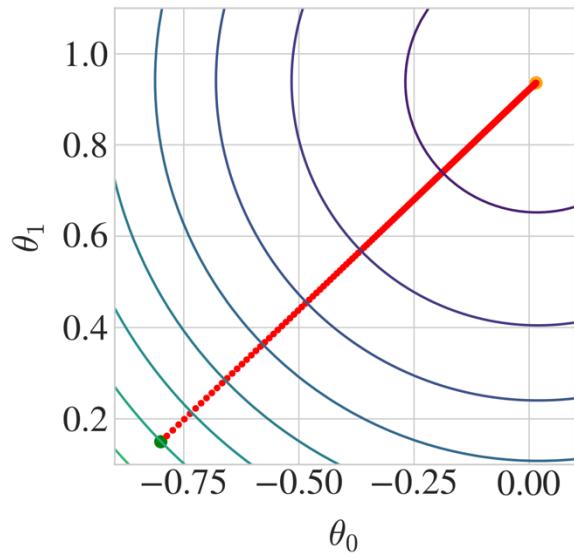
$$\gamma = 2/n$$



$$\gamma = 0.1$$



$$\gamma = 0.5$$



$$\gamma = 1$$

1

Find a research article applying machine learning on the topic of your choice

(from public databases, journals, etc.)

Examples of public databases:

- [ArXiv](#) for physics, maths, computer science, finance/economy,
- [BioRxiv](#) for biology, neuroscience, ecology,
- [ChemRxiv](#) for chemistry.

Example of journals: [Nature](#), [Science](#), whatever you want.

An example of article: Chemistry Nobel prize [Alphafold](#).

2

Write a Machine Learning **analysis** of the paper

(maximum 3 pages)

N.B: Analysis ≠ summary

1. Give context: what is the scientific question tackled, why is ML useful here,
2. Expose the problem (from an ML point of view): supervised/unsupervised framework, what data are used, how much, etc.
3. Present the model that is used to solve the problem and the optimisation process,
4. At each stage (or at the end) **discuss alternatives**: other models, other data, other problem formulation -- and **question the process**: do you see signs of overfitting, is the model well-adapted in your opinion?

3

Send me your work by email (tony.bonnaire@ens.fr). The document should mention your name and a link to the article you chose. **DUE BY NOV. 14th**