



# Machine Learning: A new toolbox for Theoretical Physics

Juan Rojo

VU Amsterdam & Theory group, Nikhef

***D-ITP Advanced Topics in Theoretical Physics***

**18/11/2019**

# Course Overview

# Schedule

- 📌 **Four lectures**, each taking place **between 11am and 1pm** at room **H331** of Nikhef
  - 📌 In the corresponding afternoons: hands-on tutorials, divided between **H331** and **H320**
- 
- ✓ Monday **18th of November** 2019: Lecture (11am to 1pm) and Tutorial (2pm to 5pm).
  - ✓ Monday **25th of November** 2019: Lecture (11am to 1pm) and Tutorial (2pm to 5pm).
  - ✓ Monday **2nd of December** 2019: Lecture (11am to 1pm) and Tutorial (2pm to 5pm).
  - ✓ Monday **9th of December** 2019: Lecture (11am to 1pm) and Tutorial (2pm to 5pm).

The course is at **full capacity**, to please make sure to be in the lecture and tutorial rooms in time to find place and to avoid disrupting the teaching

# Exam

The **marking** of this module is composed by two parts

- Writing of a **short report** (4 pages) about a specific **application of Machine Learning algorithms** to a physics problem that you find interesting or relevant for your research. This report can be written individually or in groups of at most 3 students. Including possible **code examples** is encouraged but not required

You need to send this report by **Friday 13th December**

- Presentation** of the contents of this report on **Monday 16th**: 10 min + discussion

Only students that write and present their report will obtain a **pass** for the course, provided their **quality is deemed sufficient**

The examination is only required if you aim to obtain ECs from this course!

# References

the literature on **Machine Learning and their applications to physics** is vast.  
when preparing these lectures the following resources have been used:

- A high-bias, low-variance introduction to Machine Learning for physicists*, Pankaj Mehta, Ching-Hao Wang, Alexandre G. R. Day, and Clint Richardson, Phys. Rept. 810, 1 (2019), arXiv:1803.08823 [physics.comp-ph]  
*main reference, also for the tutorials. Most examples discussed in this lectures are taken from this report*
- Machine learning and the physical sciences*, G. Carleo, I. Cirac, K. Cranmer, L. Daudet, M. Schuld, N. Tishby, L. Vogt-Maranto and L. Zdeborová, arXiv:1903.10563 [physics.comp-ph].
- Lectures on Machine Learning*, S. Carrazza, Taller de Altas Energies 2018 (TAE18), <http://benasque.org/2018tae/>
- Lectures on Artificial Intelligence, Deep Learning, Advanced Machine Learning*, G. Louppe, lecture materials and tutorials available from <https://github.com/groupppe/info8010-deep-learning>, <https://github.com/groupppe/info8004-advanced-machine-learning>

*plus many other ML resources online!*

# Tutorials

the hands-on tutorials will allow you to familiarise with the machine learning concepts presented in the lectures by means of **practical examples**

these tutorials, alongside with the rest of course materials, can be found in

**<https://github.com/juanrojochacon/ml-ditp-attp/>**

this **GitHub repository** will be updated as the course goes on, so make sure you pull frequently

the course tutorials will be based on **Python** (3.7) which hopefully most of you have experience with. These tutorials do not involve writing new code altogether but rather adapting existing programs and try new things

make sure you bring your laptop and have an **up-to-date Python installation** beforehand, e.g. from Conda or Homebrew. Test it with some of the example programs in the repo!



# Today's lecture

- ✿ Why Machine Learning? Basic concepts and terminology
- ✿ Supervised Learning: model fitting and regression
- ✿ The need for regularisation (cross-validation)
- ✿ The bias/variance tradeoff and Bayesian Inference
- ✿ Optimisers in Supervised Learning: Gradient Descent and Genetic Algorithms

*Tutorial 1: (a) Model fitting, (b) Testing Gradient Descent Methods*

# Why Machine Learning?

# Why this course?

Machine Learning rightly deserves to be part of the **toolbox of a theoretical physicist**

- ✿ Essential for building **modern models and algorithms** in various areas of physics
- ✿ Very **fast developments** both in algorithms and in computing platforms have significantly extended the breadth of problems that can be tackled with ML
- ✿ Deep **physical connections** with many problems in theoretical physics, e.g. quantum computation
- ✿ Applied to problems even in very **formal fields** e.g. string theory
- ✿ Large interested in the community, **societal implications** (AI hype)

Furthermore, expertise in ML/AI is powerful asset for also for careers outside academia

# AI vs ML

Machine Learning is part of the **Artificial Intelligence paradigm**

*AI is the science and engineering of making intelligent machines (McCarthy '56)*

## ARTIFICIAL INTELLIGENCE



Turing Test Devised  
1950

1960s

ELIZA  
1964 - 1966

Edward Shortliffe writes MYCIN,  
an Expert or Rule based System,  
to classify blood disease  
1970s

## MACHINE LEARNING



1980s

1990s

IBM Deep Blue defeats Grand  
Master Garry Kasparov in chess  
1996

2000s

ImageNet Feeds  
Deep Learning  
2009

AlphaGo defeats Go  
champion Lee Sedol  
2016



## DEEP LEARNING

# AI vs ML

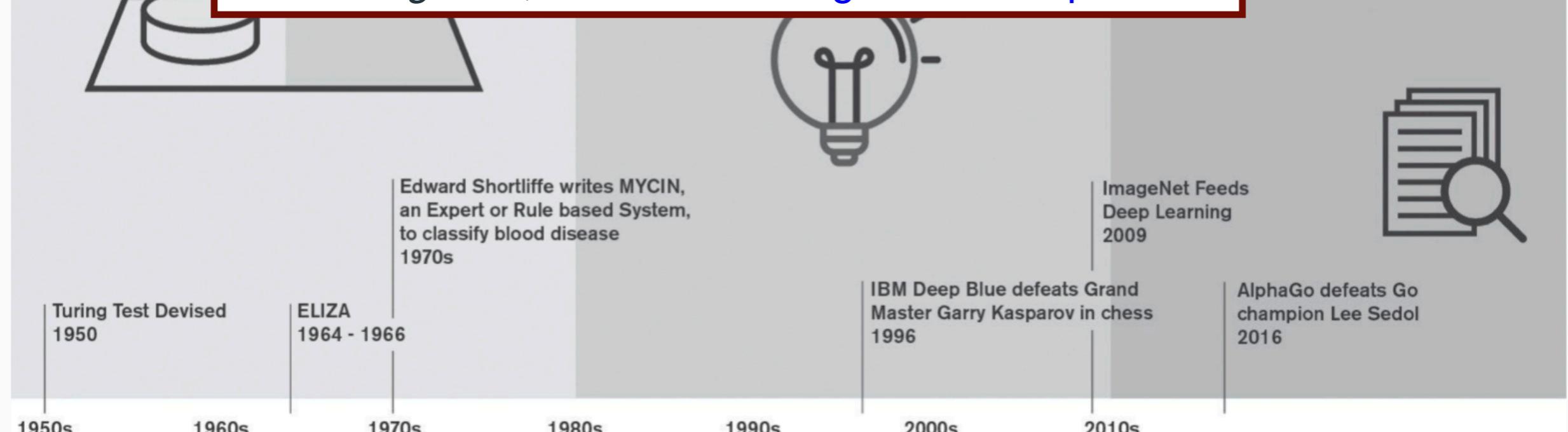
Machine Learning is part of the **Artificial Intelligence paradigm**

*AI is the science and engineering of making intelligent machines (McCarthy '56)*

## ARTIFICIAL INTELLIGENCE

## MACHINE LEARNING

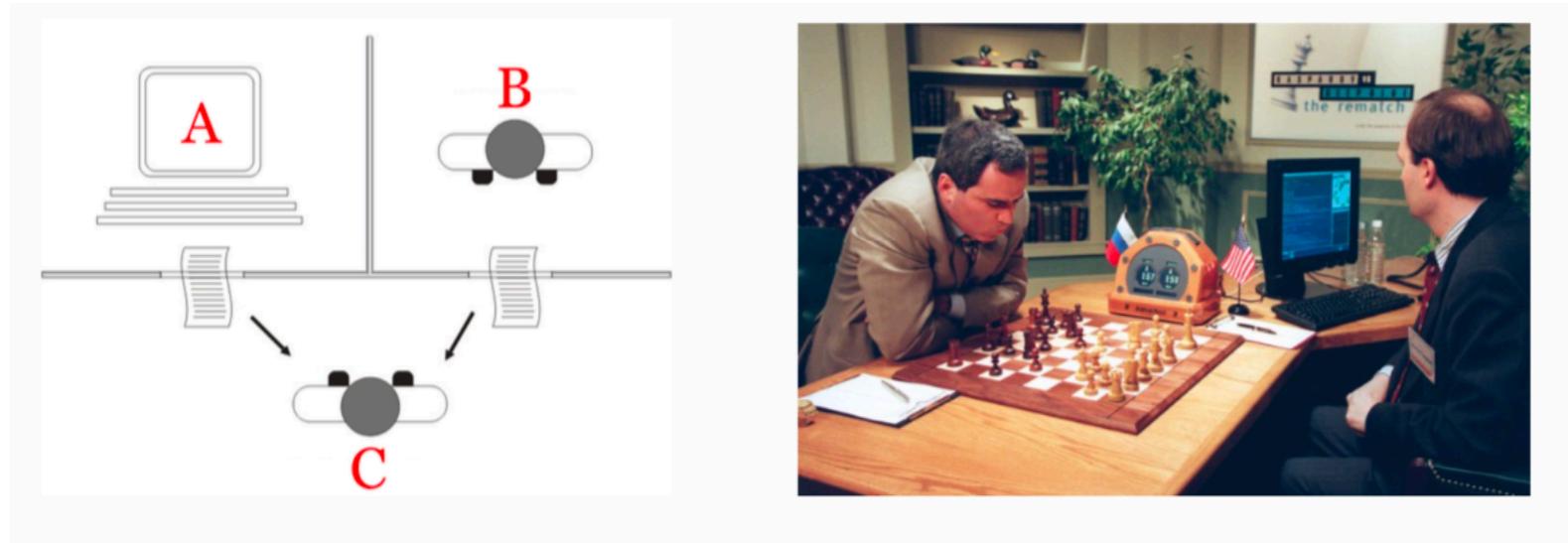
A.I. consist in the development of **computer systems** to perform tasks commonly associated with *intelligence*, such as **learning from examples**



# Problems in AI

Most problems tackled with Artificial Intelligence fall in **two categories**

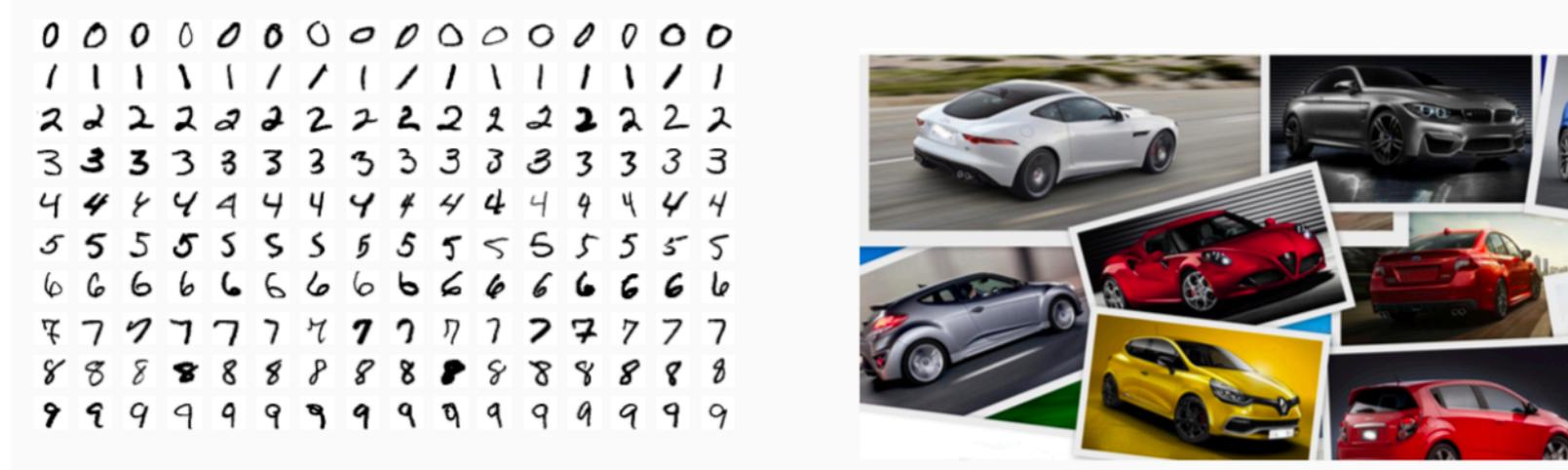
- (1) abstract and formal: *easy for computers but difficult for humans*  
**knowledge-based approach**



e.g. *chess (DeepBlue)*

*(chess is deterministic game with finite number of options )*

- (2) intuitive, hard to formalize: *easy for humans but difficult for machines*  
**concept capture and generalisation**



e.g. *pattern recognition*

# Problems in AI

Most problems tackled with Artificial Intelligence fall in **two categories**

- (1) To excel in tasks which are intuitive to humans but difficult to machines, an A.I. system needs to **acquire its own knowledge**: the Machine is Learning

*chess (DeepBlue)*

Machine Learning algorithms allow computers the ability to **carry out a task without being explicitly programmed how to do it by learning from examples**

- (2) intuitive, hard to formalize: *easy for humans but difficult for machines*  
**concept capture and generalisation**

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8
9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9



*e.g. pattern  
recognition*

# ML is everywhere

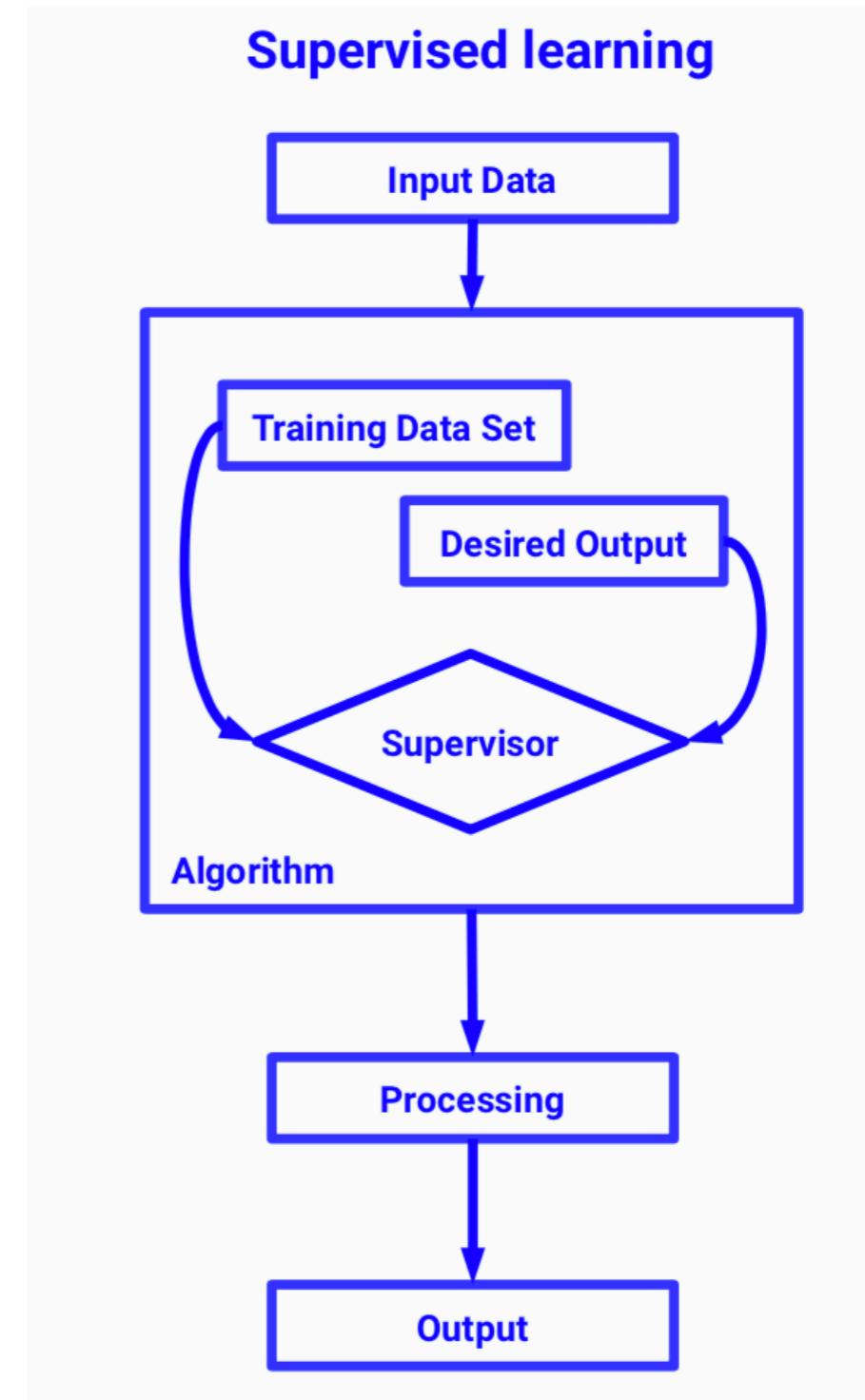
- **Database mining:**
  - Search engines
  - Spam filters
  - Medical and biological records
- **Intuitive tasks for humans:**
  - Autonomous driving
  - Natural language processing
  - Robotics (reinforcement learning)
  - Game playing (DQN algorithms)
- **Human learning:**
  - Concept/human recognition
  - Computer vision
  - Product recommendation



# Learning to learn

**Machine Learning** algorithms can be divided into several classes, including

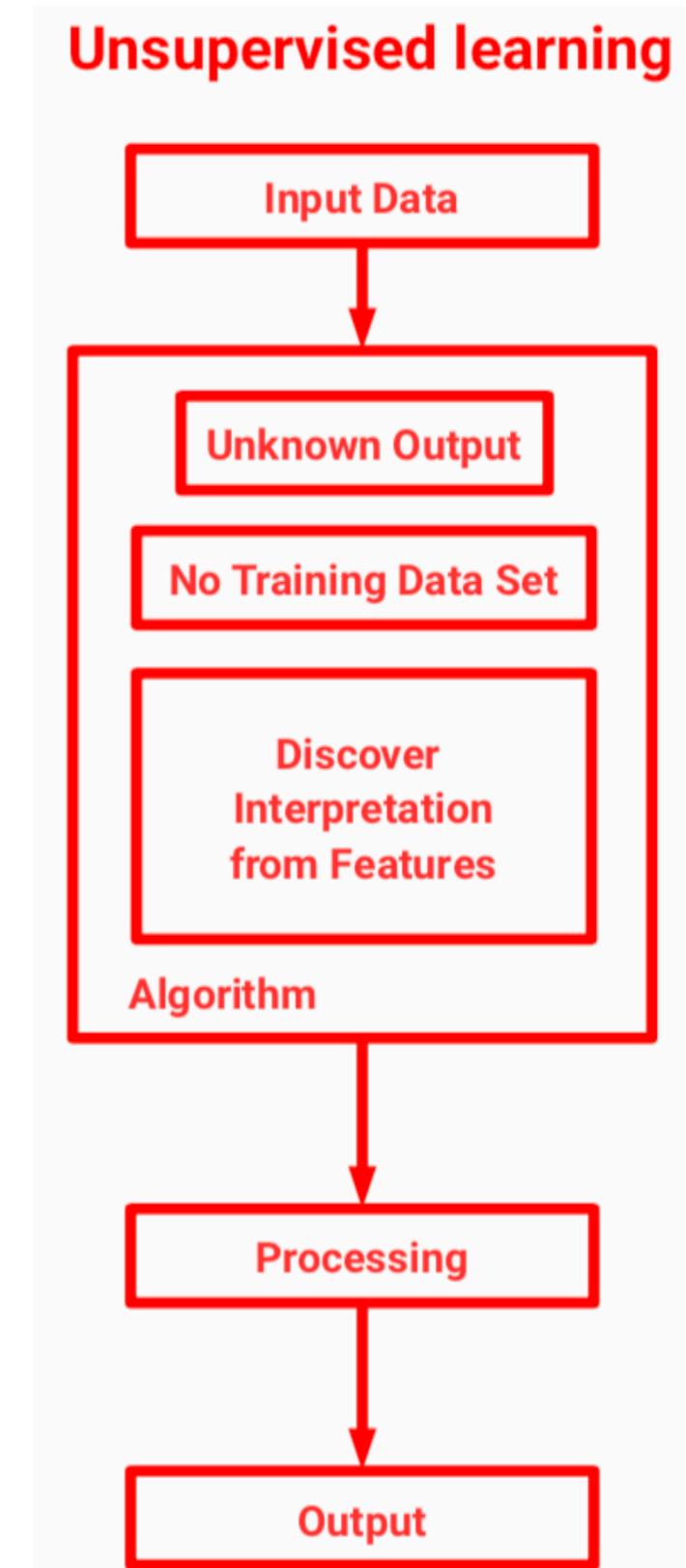
- ✿ **Supervised Learning:** regression, classification, ...



# Learning to learn

**Machine Learning** algorithms can be divided into several classes, including

- ⌚ **Supervised Learning:**  
regression, classification, ...
- ⌚ **Unsupervised Learning:**  
clustering, data dimensional reduction, ....



# Learning to learn

**Machine Learning** algorithms can be divided into several classes, including

- ⌚ **Supervised Learning:**  
regression, classification, ...
- ⌚ **Unsupervised Learning:**  
clustering, data dimensional reduction, ....
- ⌚ **Reinforcement learning:**  
efficiently react to changing environment



# The Machine Learning Galaxy I



*In this course we will have time to cover only subset of ML algorithms and applications!*

# The Machine Learning Galaxy II

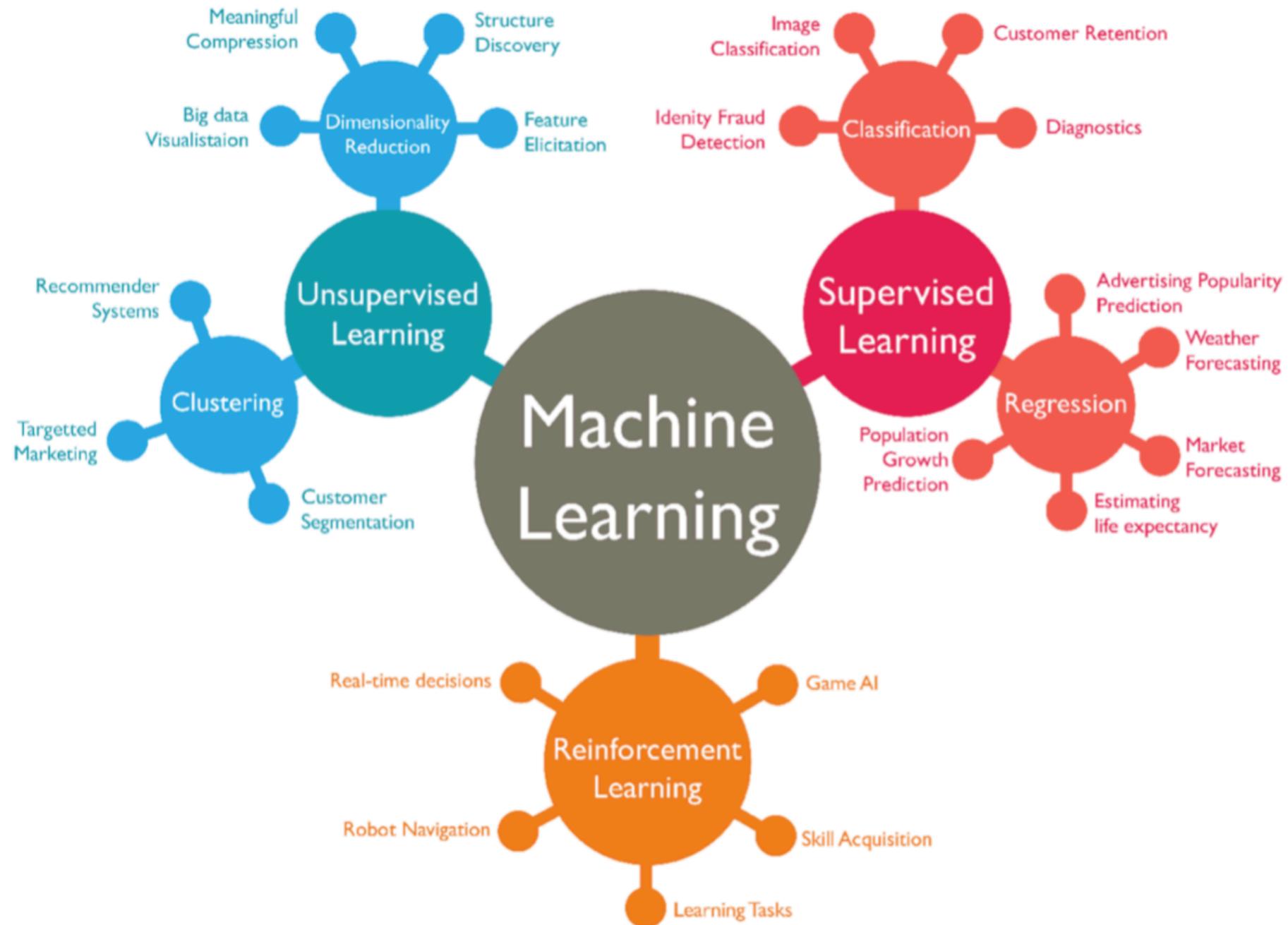


Image via [Abdul Rahid](#)

*In this course we will have time to cover only subset of ML algorithms and applications!*

# **Supervised Learning: Model Fitting and Regression**

# Supervised learning

We denote as **supervised learning** the ML task of **learning a function** that maps a **vector of inputs** to a **vector of outputs** from a finite set of training example

note that some assumptions will be needed: a function is an **infinite-dimensional object** but learning takes place from a **finite number of examples**

main property of supervised learning: the **training samples are labeled**

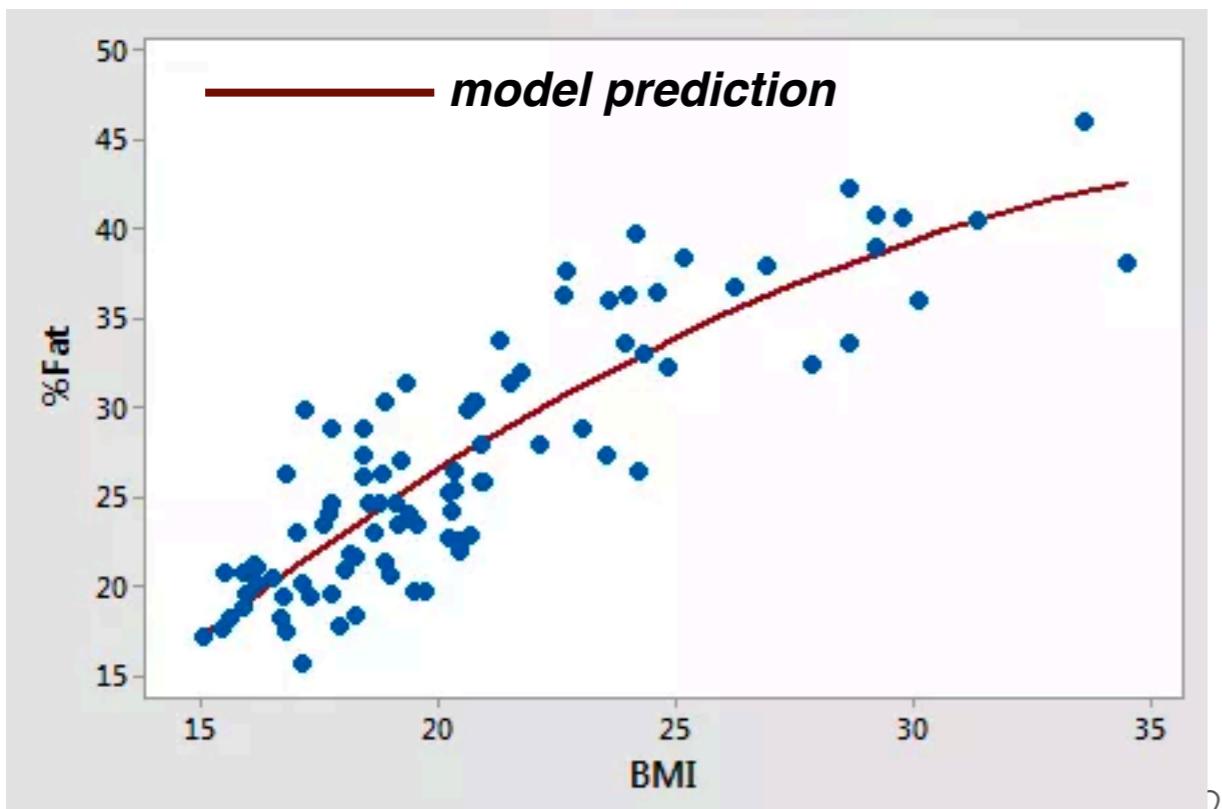
# Supervised learning

We denote as **supervised learning** the ML task of **learning a function** that maps a **vector of inputs** to a **vector of outputs** from a finite set of training example

note that some assumptions will be needed: a function is an **infinite-dimensional object** but learning takes place from a **finite number of examples**

main property of supervised learning: the **training samples are labeled**

*continuous outputs:  
regression*



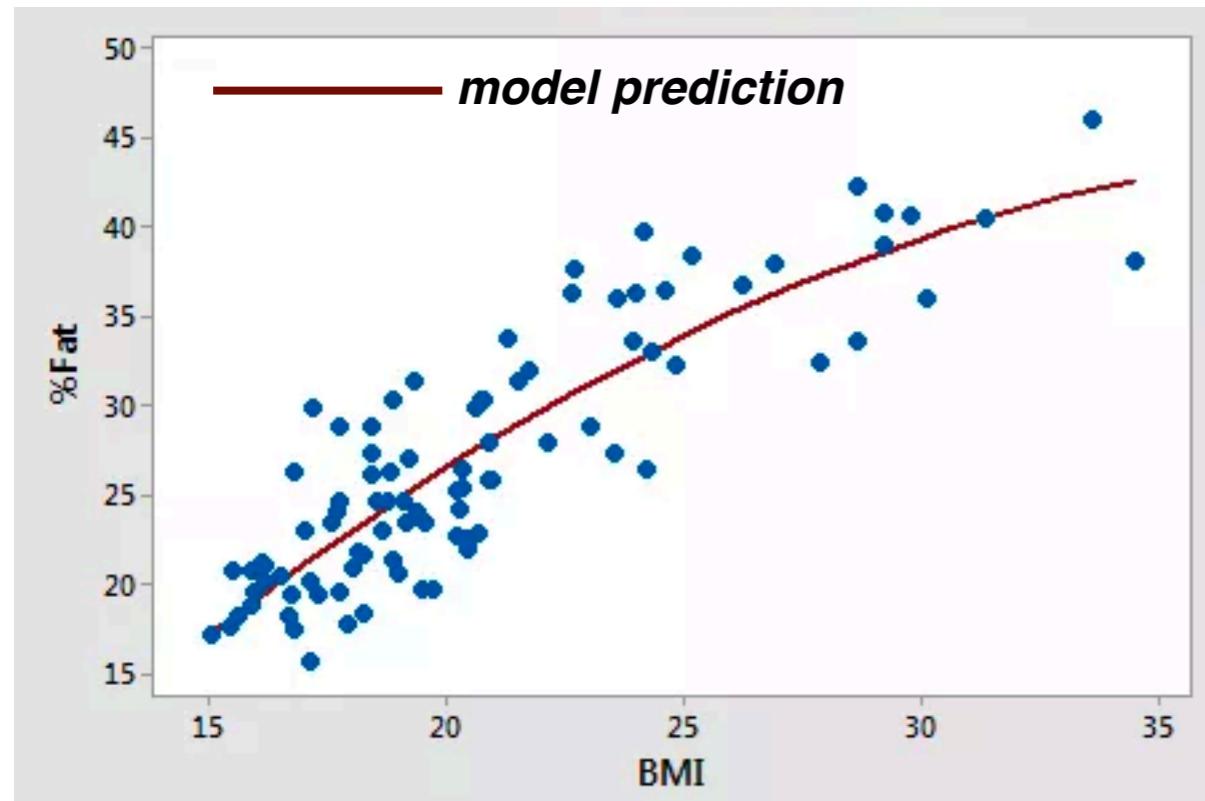
# Supervised learning

We denote as **supervised learning** the ML task of **learning a function** that maps a **vector of inputs** to a **vector of outputs** from a finite set of training example

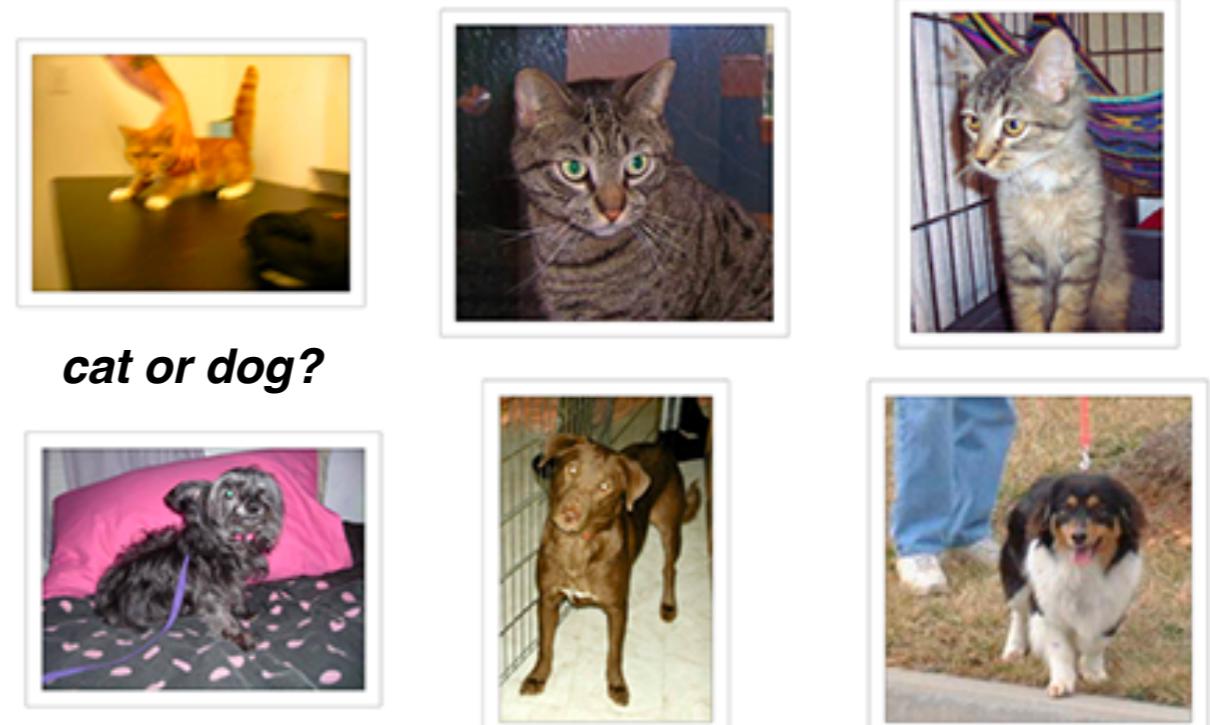
note that some assumptions will be needed: a function is an **infinite-dimensional object** but learning takes place from a **finite number of examples**

main property of supervised learning: the **training samples are labeled**

*continuous outputs:  
regression*



*discrete outputs:  
classification*



# Setting up the problem

problems in **Supervised Machine Learning** are defined by the following ingredients:

(1) **Input dataset:**  $\mathcal{D} = (\mathbf{X}, \mathbf{Y})$

array of **independent**  
variables

array of **dependent**  
variables

$$\mathbf{X} = (x_1, x_2, \dots, x_N)$$

$$\mathbf{Y} = (y_1, y_2, \dots, y_M)$$

(2) **Model:**

$$f(\mathbf{X}, \theta)$$

mapping between dependent  
and independent variables

$$f : \mathbf{X} \rightarrow \mathbf{Y}$$

model parameters

$$\theta = (\theta_1, \theta_2, \dots, \theta_P)$$

# Setting up the problem

problems in **Supervised Machine Learning** are defined by the following ingredients:

**(1) Input dataset:**  $\mathcal{D} = (\mathbf{X}, \mathbf{Y})$

**(2) Model:**  $f(\mathbf{X}, \theta)$

**(3) Cost function:**  $C(\mathbf{Y}; f(\mathbf{X}; \theta))$

The cost function measures how well the model (for a specific choice of its parameters) is able to describe the input dataset

*example of cost function for single dependent variable: sum of residuals squared*

$$C(\mathbf{Y}; f(\mathbf{X}; \theta)) = \chi^2 = \frac{1}{n} \sum (y_i - f(x_i; \theta))^2$$

# Setting up the problem

problems in **Supervised Machine Learning** are defined by the following ingredients:

(1) **Input dataset:**  $\mathcal{D} = (\mathbf{X}, \mathbf{Y})$

(2) **Model:**  $f(\mathbf{X}, \theta)$

(3) **Cost function:**  $C(\mathbf{Y}; f(\mathbf{X}; \theta))$

The cost function measures how well the model (for a specific choice of its parameters) is able to describe the input dataset

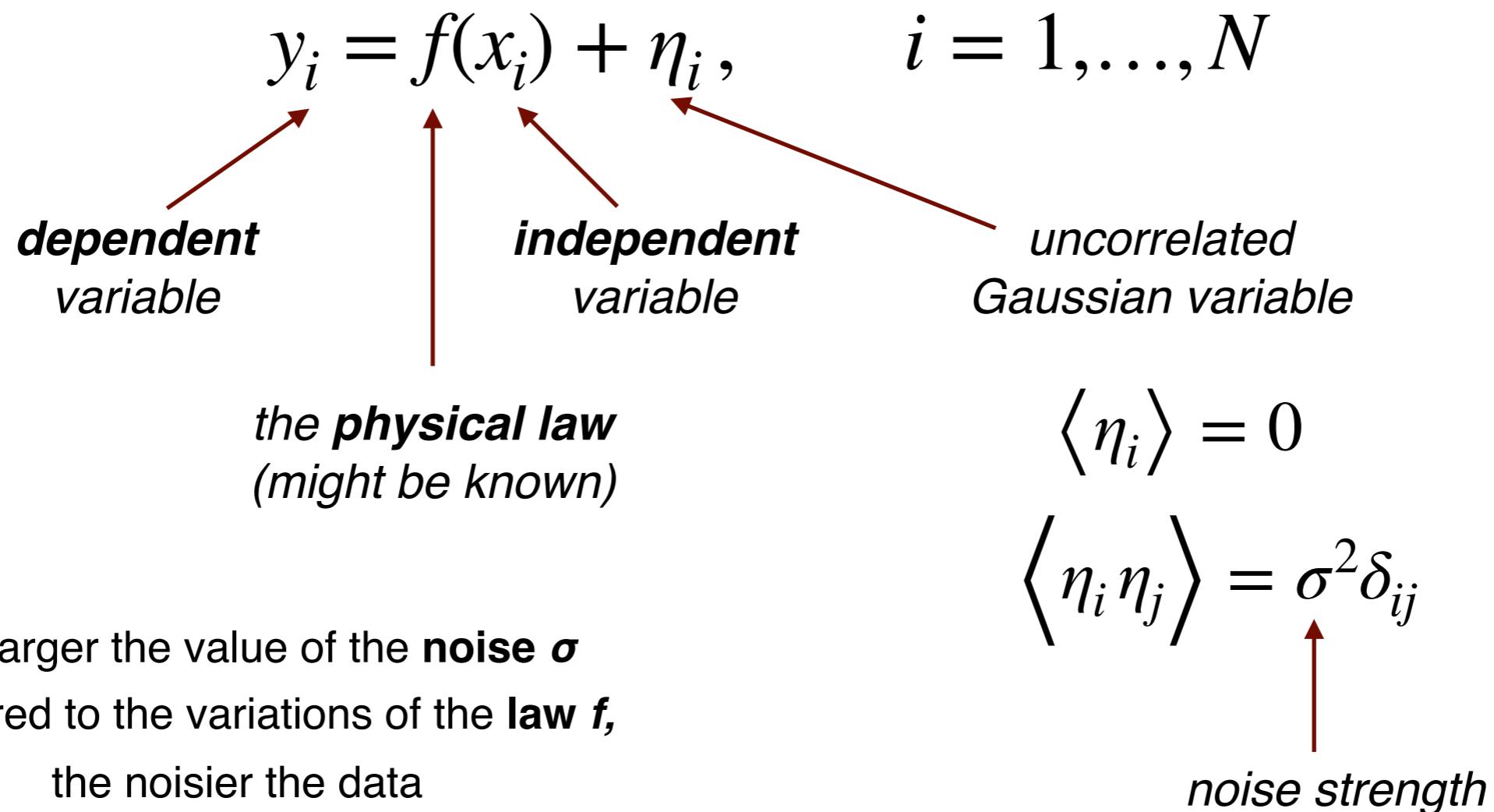
Fitting the model means determining the values of its parameters which **minimise the cost function**

$$\frac{\partial C(\mathbf{Y}; f(\mathbf{X}; \theta))}{\partial \theta_i} \Bigg|_{\theta=\theta_{\text{opt}}} = 0$$

# Model fitting

Let's illustrate why cross-validation is a central ingredient for Machine Learning applications with a simple example: polynomial regression in 1D

First of all we can generate data following a **known model** and then adding **stochastic noise**, to emulate a realistic situation



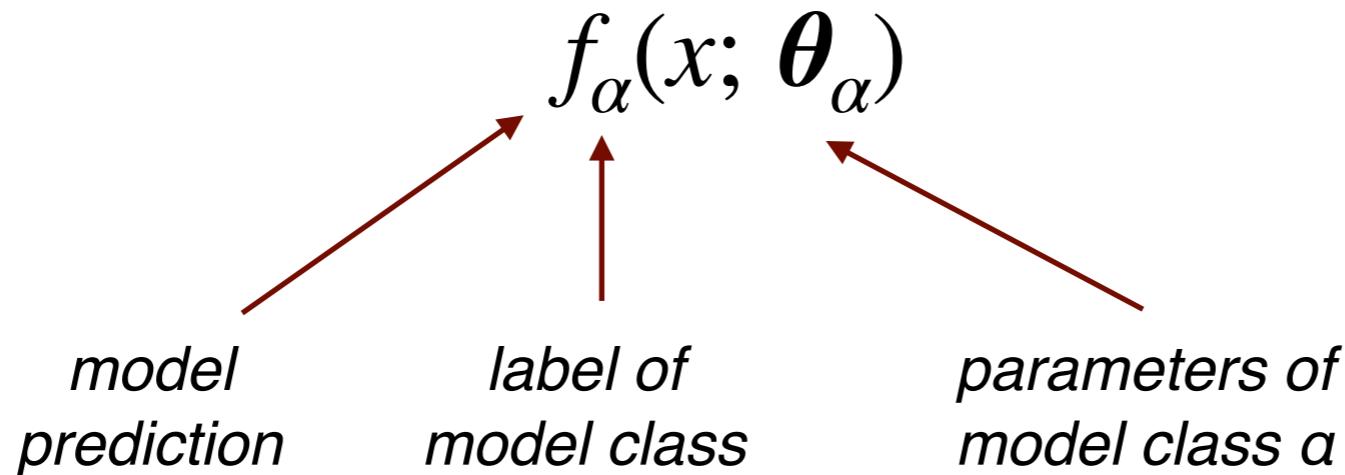
# Model fitting

Now in the real world we would have only the information about the ``measurements''

$$\mathcal{D} = (\mathbf{X}, \mathbf{Y})$$

and our goal is to extract the **underlying physical law** from this data

We need to define **classes of models** that might provide a good description of the data, in a way that its parameters encode the main features of the physical law



we want to compare the performance of models with **different complexities**: different functional forms, number of parameters, intrinsic flexibility ....

# Model fitting

The simplest possible model is a polynomial: **polynomial regression**

$$f_\alpha(x; \theta_\alpha) = \sum_{j=0}^{N_\alpha} \theta_{\alpha,j} x^j$$

for example the model class that contains all possible cubic polynomials is

$$f_3(x; \theta_3) = \sum_{j=0}^3 \theta_{3,j} x^j$$

a more complex model does not necessarily imply a more predictive one: the appropriate amount of complexity depends on the **features of the data sample** (eg size, variability)

in linear regression the model parameter are found analytically with **least-squares method**

$$\hat{\theta} = \arg \min_{\theta} \left\{ \sum_{i=1}^N (y_i - f_\alpha(x_i; \theta_\alpha))^2 \right\}$$

*minimise sum of residuals squared*

# Model fitting

Simples case: **least-squares method** for order-one polynomials

$$\frac{\partial}{\partial \theta_0} \sum_{i=1}^N (y_i - \theta_0 - \theta_1 x_i)^2 \Big|_{\widehat{\theta}_0} = 0$$

$$\frac{\partial}{\partial \theta_1} \sum_{i=1}^N (y_i - \theta_0 - \theta_1 x_i)^2 \Big|_{\widehat{\theta}_1} = 0$$

and by solving this linear system of equations one obtains the **model parameters**

$$\widehat{\theta}_1 = \frac{\langle xy \rangle - \langle x \rangle \langle y \rangle}{\langle x^2 \rangle - \langle x \rangle^2}$$

$$\widehat{\theta}_0 = \langle y \rangle - \widehat{\theta}_1 \langle x \rangle$$

*averages computed over the N elements of the data sample*

# Regularisation

What is the best strategy to **determine the model parameters**?

Seems a silly question, surely those are simply **minimum of cost function**?

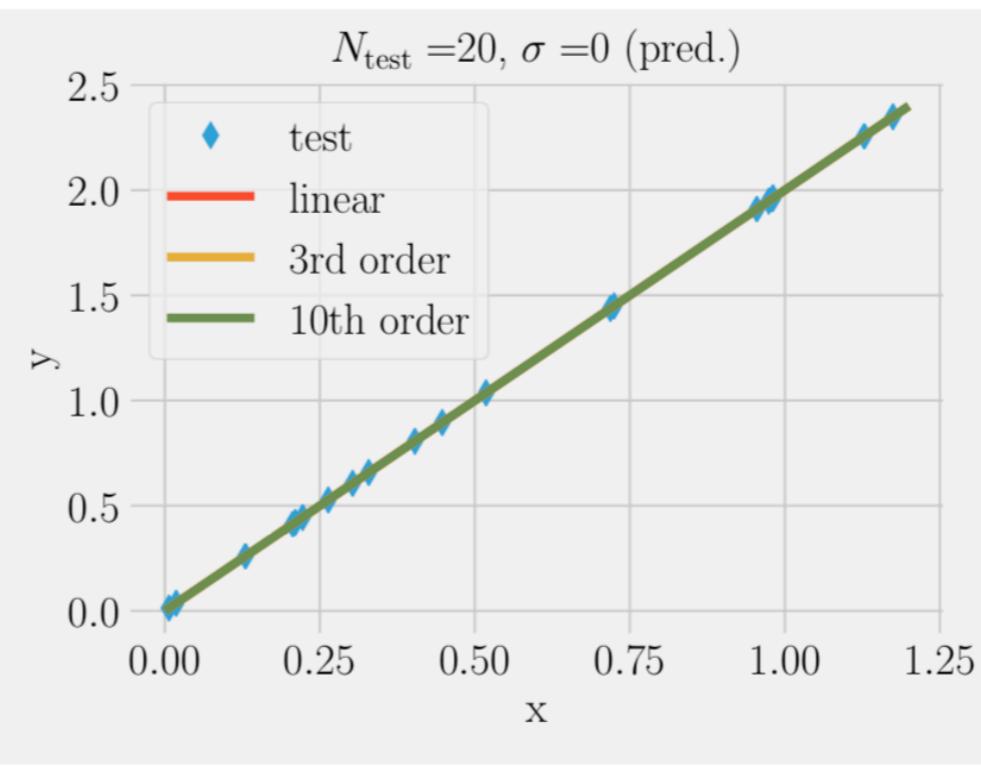
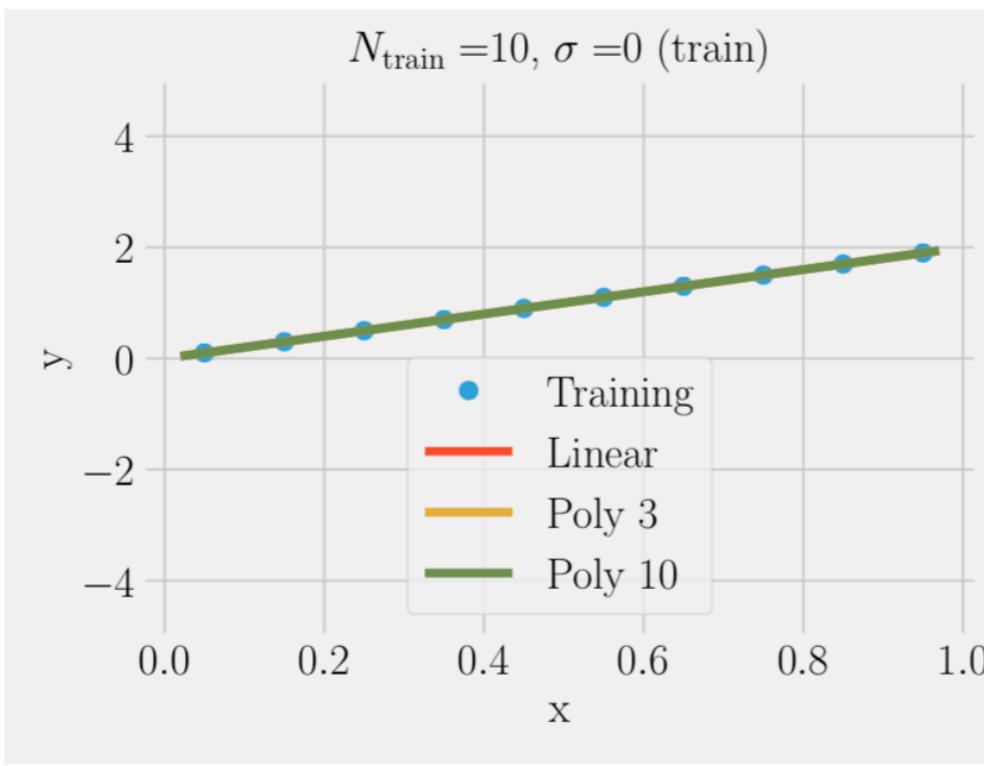
$$\hat{\theta} = \arg \min_{\theta} \left\{ \sum_{i=1}^N (y_i - f_{\alpha}(x_i; \theta_{\alpha}))^2 \right\}$$

However this is in general **not the case**, because:

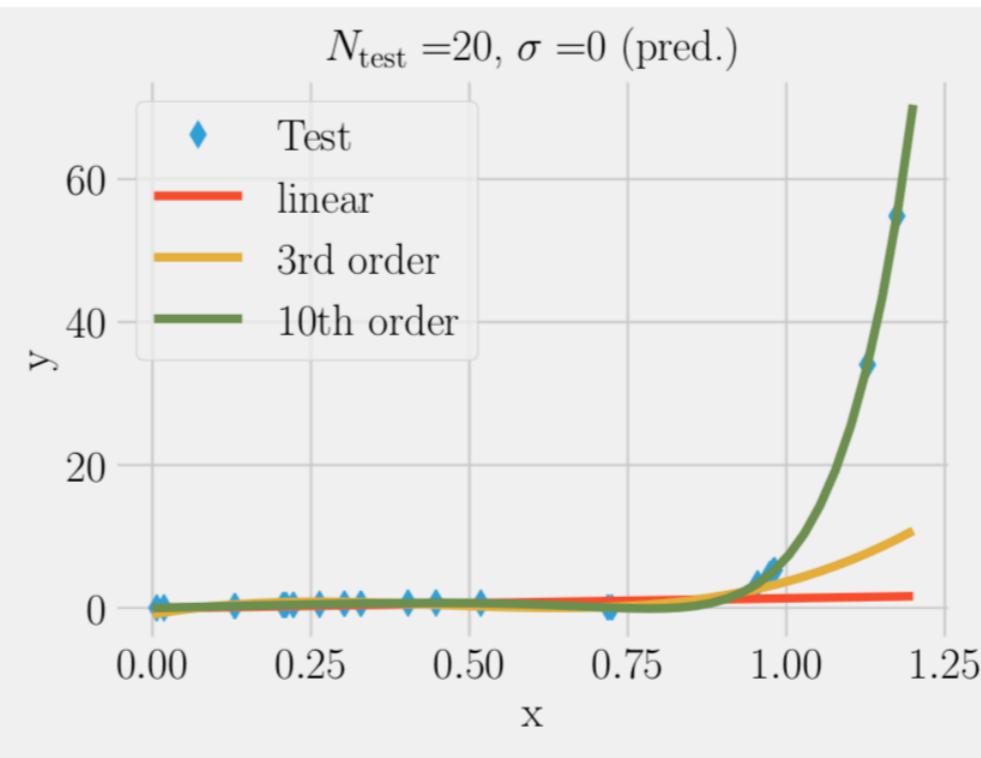
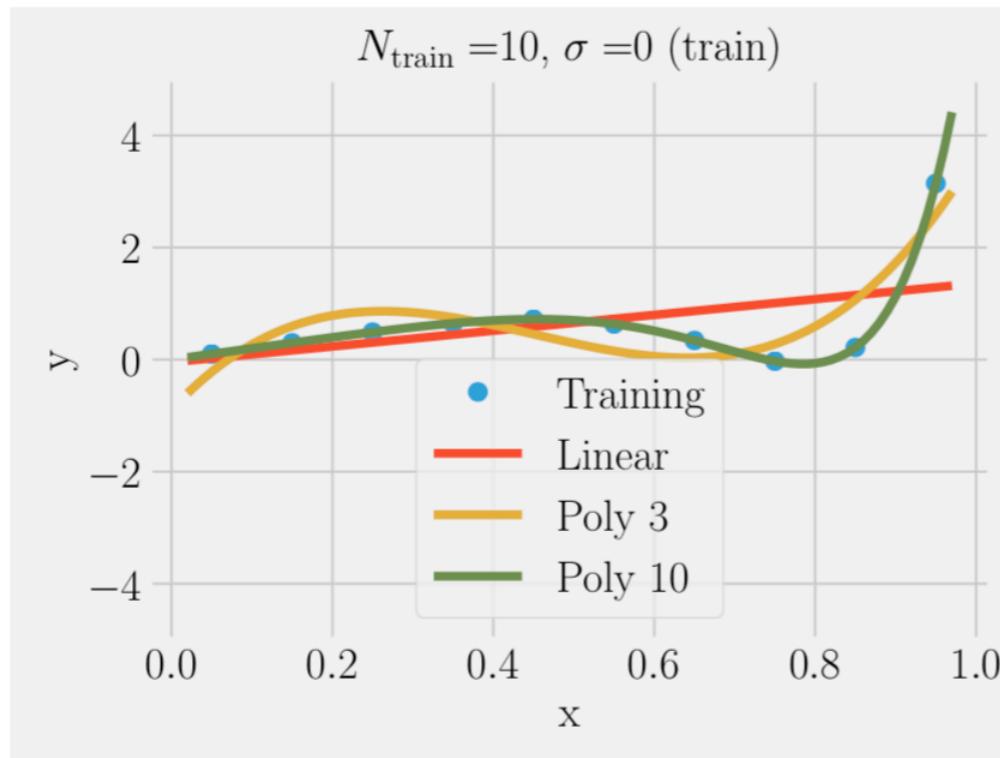
- ➊ Real world data is **noisy**: we want to learn the underlying law, not the statistical fluctuations
- ➋ More than fitting the data, our real goal is to create a model that **predicts future/different data**: we need figures of merit outside the training dataset
- ➌ To ensure that our model describes the underlying law (and thus one can safely generalise) rather than the noise, a regularisation procedure needs to be used

# Model fitting

$$f(x) = 2x, \quad x \in [0,1]$$

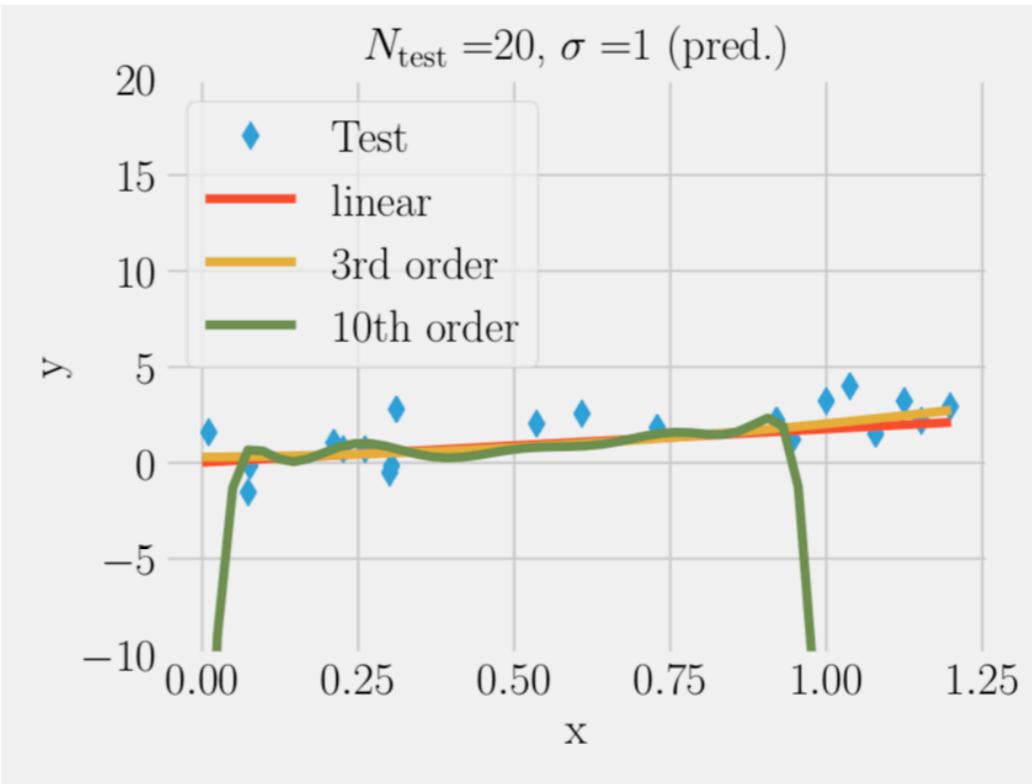
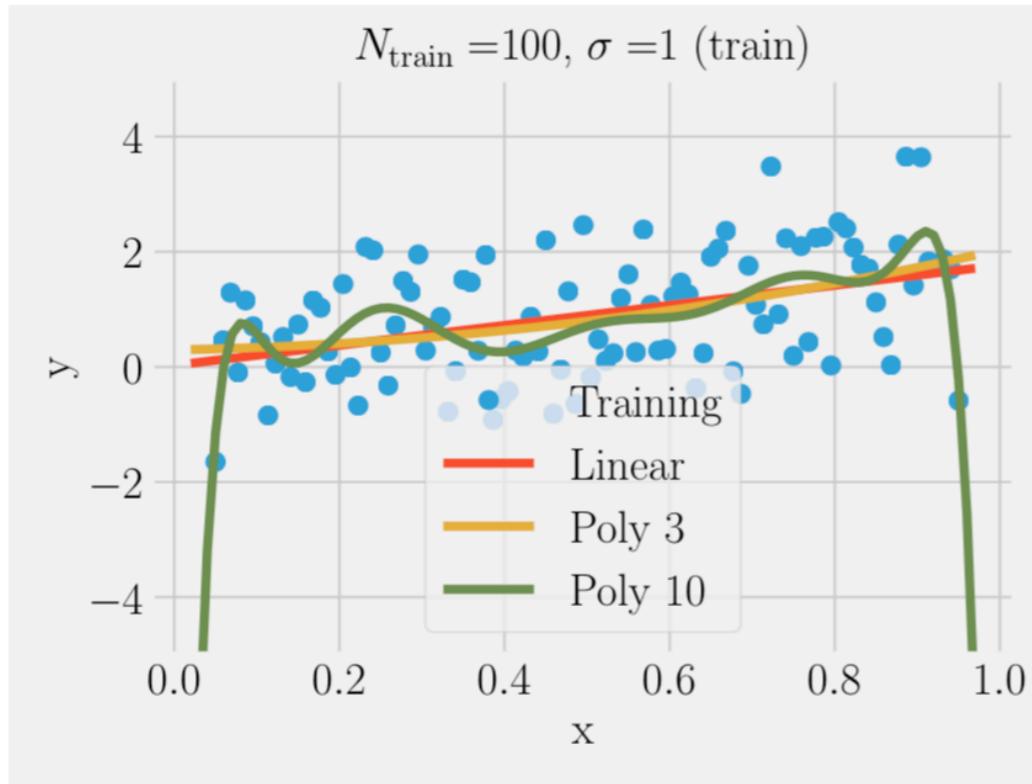


$$f(x) = 2x - 10x^5 + 15x^{10}, \quad x \in [0,1]$$

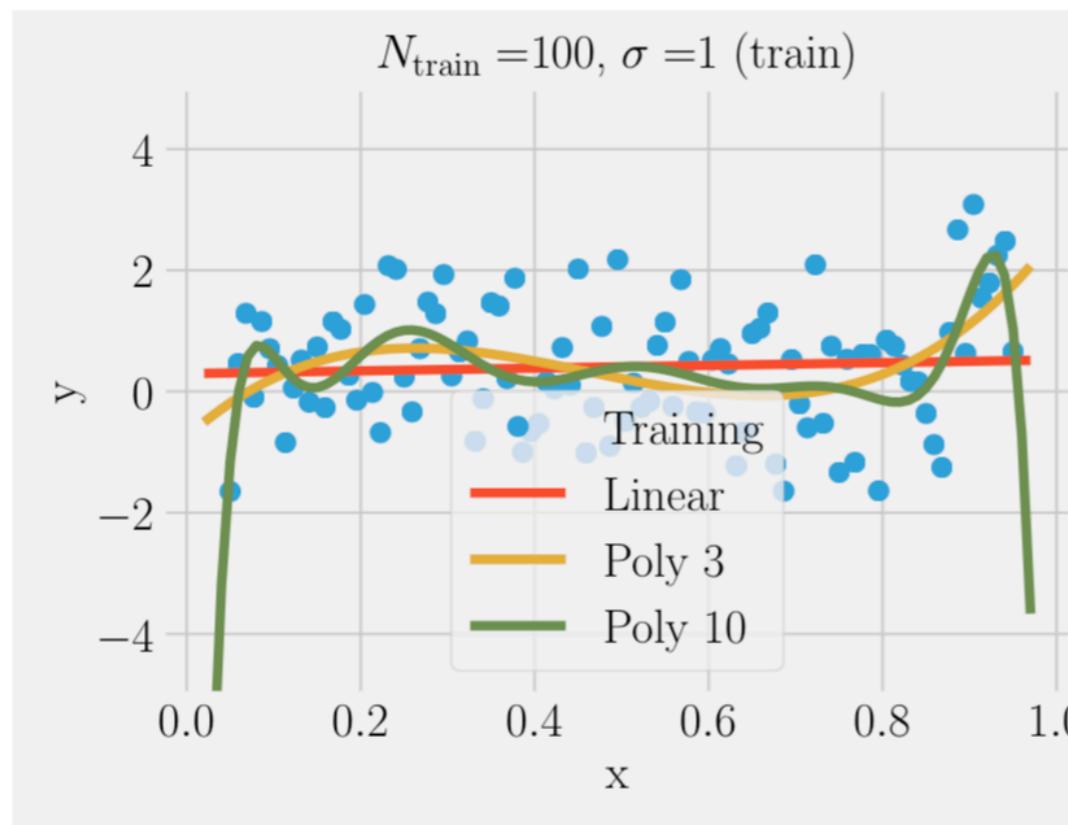


# Model fitting

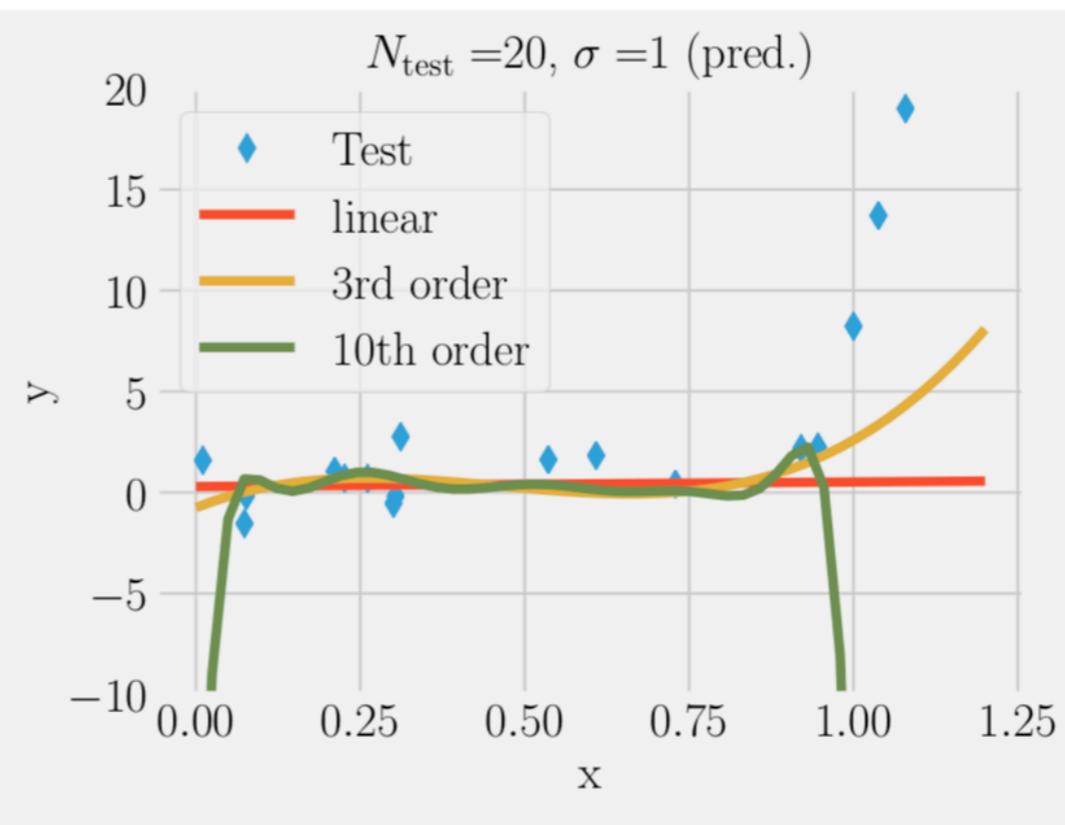
$$f(x) = 2x, \quad x \in [0,1]$$



$$f(x) = 2x - 10x^5 + 15x^{10}, \quad x \in [0,1]$$



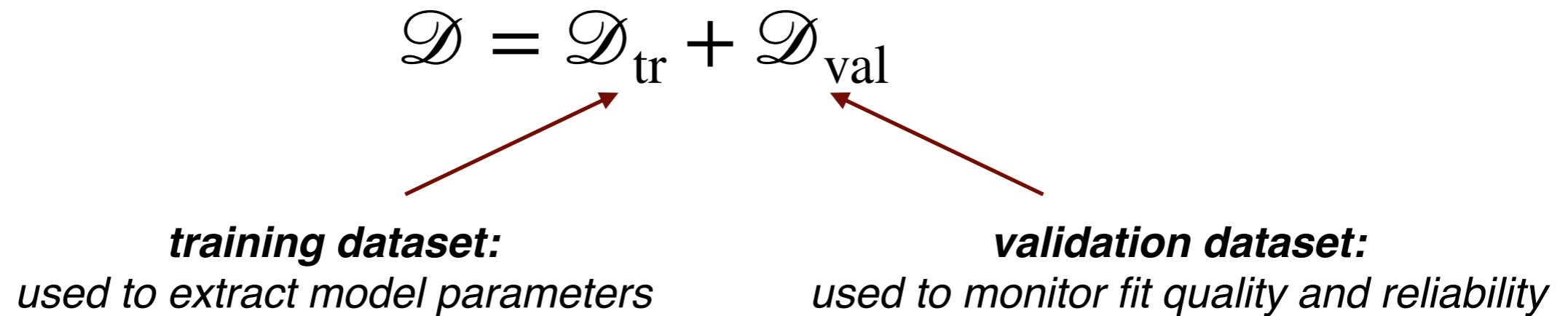
*in the presence of noise, models with less complexity can exhibit improved predictive power*



# Cross-validation (regularisation)

What is the best strategy to determine the model parameters?

first of all, divide input dataset into **two disjoint sets**



the model parameters are determined using information from training dataset

$$\hat{\theta} = \arg \min_{\theta} \{ C \left( \mathbf{Y}_{\text{th}}, f(\mathbf{X}_{\text{tr}}; \theta) \right) \}$$

while the performance of the model is evaluated by computing the cost function  
on the validation dataset

$$C \left( \mathbf{Y}_{\text{val}}, f(\mathbf{X}_{\text{val}}; \hat{\theta}) \right)$$

# Cross-validation (regularisation)

In-sample (training) error  $\longrightarrow E_{\text{tr}} \equiv C \left( \mathbf{Y}_{\text{tr}}, f(\mathbf{X}_{\text{tr}}; \hat{\theta}) \right)$

Out-of-sample (validation) error  $\longrightarrow E_{\text{val}} \equiv C \left( \mathbf{Y}_{\text{val}}, f(\mathbf{X}_{\text{val}}; \hat{\theta}) \right)$

Splitting the data into mutually exclusive training and validation sets provides an unbiased estimate for the **predictive performance of the model**

In ML problems one selects the model that **minimises** the out-of-sample error  $E_{\text{val}}$ , since this is the model that generalises in the most efficient way

# Cross-validation (regularisation)

In-sample (training) error  $\longrightarrow E_{\text{tr}} \equiv C \left( \mathbf{Y}_{\text{tr}}, f(\mathbf{X}_{\text{tr}}; \hat{\theta}) \right)$

Out-of-sample (validation) error  $\longrightarrow E_{\text{val}} \equiv C \left( \mathbf{Y}_{\text{val}}, f(\mathbf{X}_{\text{val}}; \hat{\theta}) \right)$

Splitting the data into mutually exclusive training and validation sets provides an unbiased estimate for the **predictive performance of the model**

In ML problems one selects the model that **minimises** the out-of-sample error  $E_{\text{val}}$ , since this is the model that generalises in the most efficient way

***Fitting is not predicting:*** in general the model that describes better a given set of data will not be the one that generalises and predicts better related datasets

# Regularisation

In ML, the common goal of regularisation strategies is to modify the learning algorithms to **reduce its generalisation error without increasing its training error**

$$C(\theta) = \frac{1}{N} \sum_{i=1}^N (y_i - f(x_i; \theta))^2 \quad \textit{unregularised cost function}$$

$$C(\theta) = \frac{1}{N} \sum_{i=1}^N (y_i - f(x_i; \theta))^2 + \lambda \theta^T \theta \quad \textit{regularised cost function}$$

*regularisation hyperparameter,  
to be tuned*

*weight-decay regularisation:  
model parameters with smaller  
L2-norms are preferred*

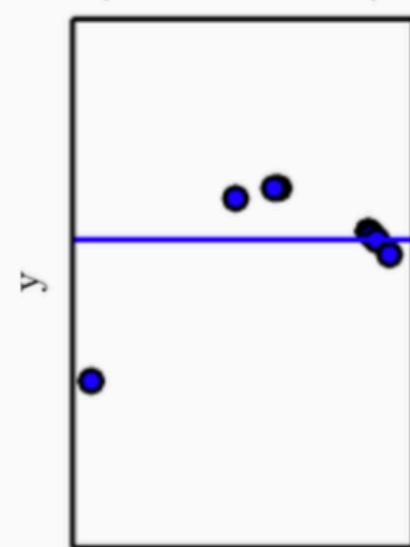
# Regularisation

In ML, the common goal of regularisation strategies is to modify the learning algorithms to **reduce its generalisation error without increasing its training error**

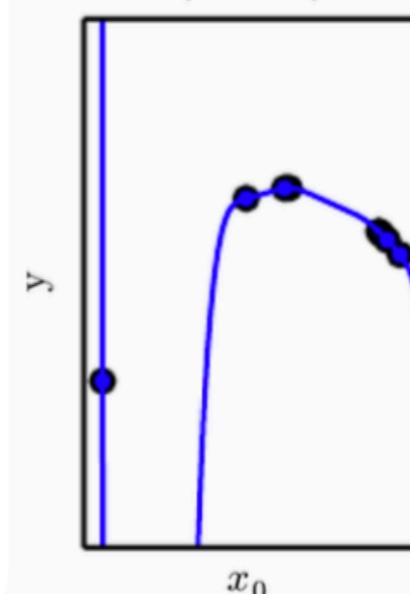
$$C(\theta) = \frac{1}{N} \sum_{i=1}^N (y_i - f(x_i; \theta))^2 \quad \text{unregularised cost function}$$

$$C(\theta) = \frac{1}{N} \sum_{i=1}^N (y_i - f(x_i; \theta))^2 + \lambda \theta^T \theta \quad \text{regularised cost function}$$

Underfitting  
(Excessive  $\lambda$ )



Overfitting  
( $\lambda \rightarrow 0$ )



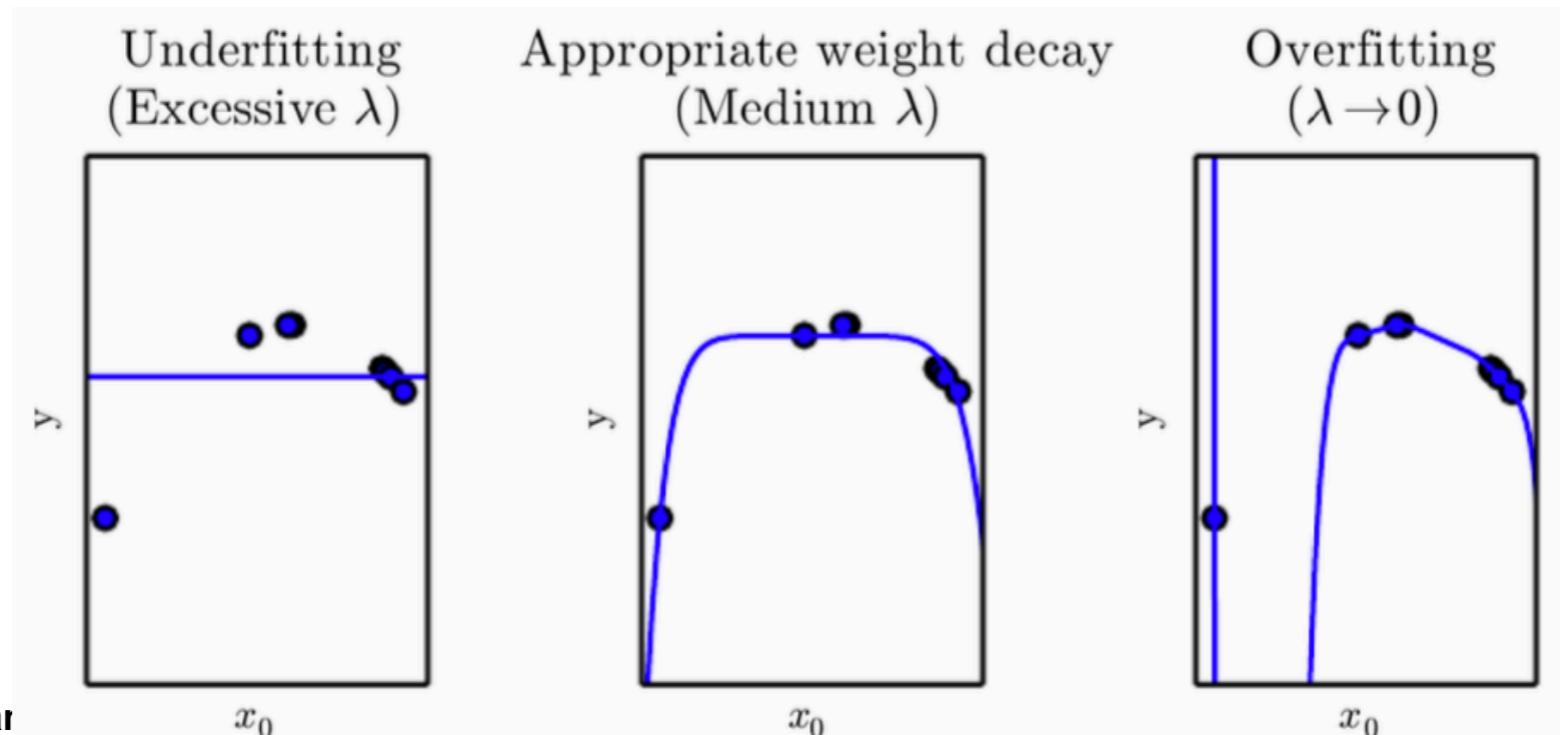
*no free lunch,  
careful tuning of  
regularisation  
required*

# Regularisation

In ML, the common goal of regularisation strategies is to modify the learning algorithms to **reduce its generalisation error without increasing its training error**

$$C(\theta) = \frac{1}{N} \sum_{i=1}^N (y_i - f(x_i; \theta))^2 \quad \text{unregularised cost function}$$

$$C(\theta) = \frac{1}{N} \sum_{i=1}^N (y_i - f(x_i; \theta))^2 + \lambda \theta^T \theta \quad \text{regularised cost function}$$

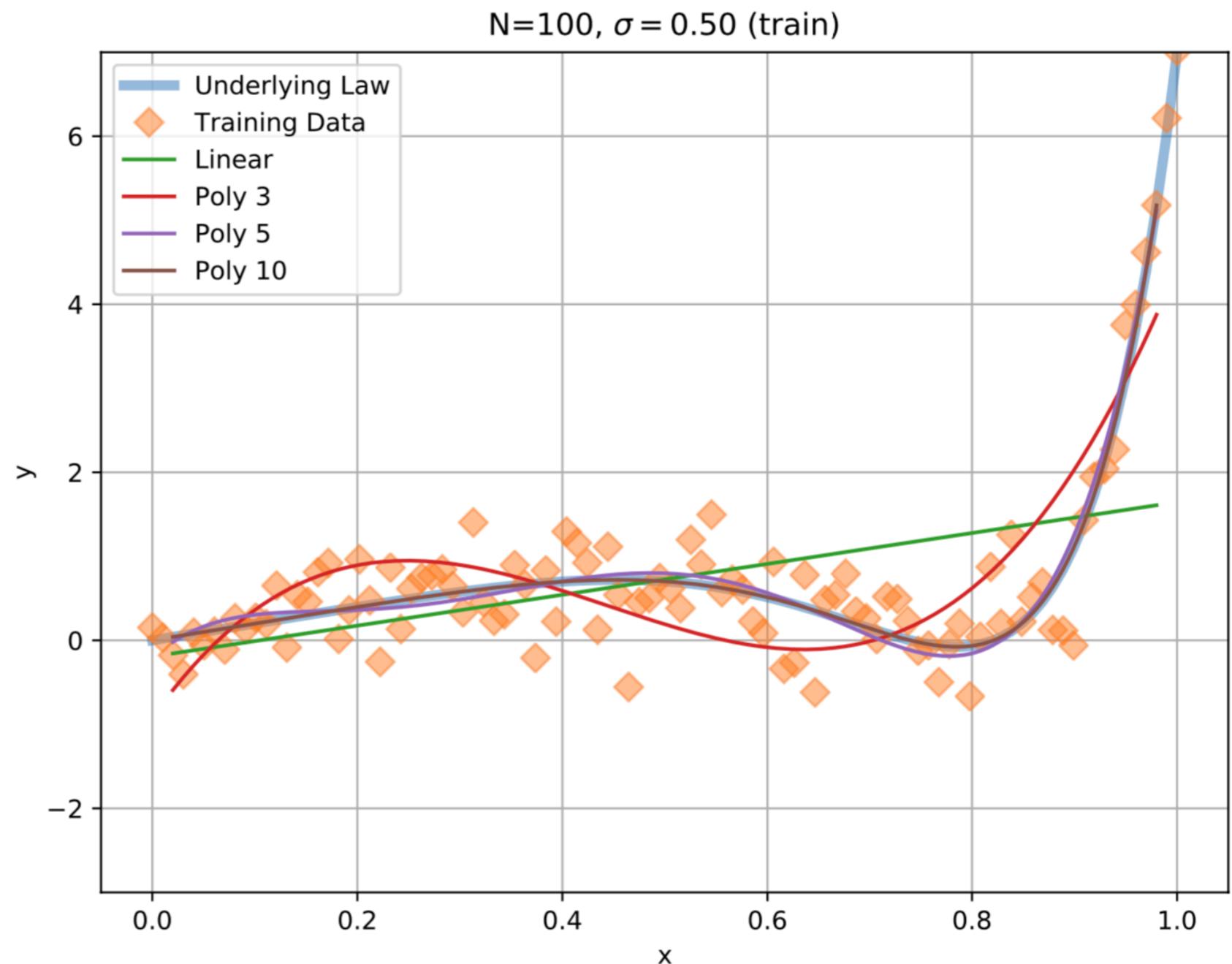


# Exercise I

starting point is the **Python script** that you will find in

<https://github.com/juanrojochacon/ml-ditp-attp/blob/master/Tutorials/Tutorial1/Lecture1-ModelFitting.py>

- Study how the cost function to the training data depends on *i*) the complexity of the underlying law, *ii*) the flexibility of the model, *iii*) the number of training points
- Now generate a validation dataset, either within the training data range [0,1] or outside it. Study which model is more predictive as a function of the same parameters as before in the two cases
- Can you **identify overfitting**? Can you think of a way of avoiding it?



# Why ML is difficult

- **Fitting existing data** is conceptually different from **making predictions about new data**
- Increasing the model complexity can improve the description of the training data but **reduce the predictive power** of the model due to overfitting, unless a suitable regularisation strategy is implemented
- For complex and/or small datasets, simple models can be better at prediction than complex models. The **“right” amount of complexity** cannot in general be determined from first principles
- It is difficult to **generalise** beyond the situations encountered in the training set: the model cannot learn what it has not seen
- Many problems that are **approachable in principle** can become **unfeasible in practice**, e.g. due to computational limitations, lack of convergence, instability

# **intermezzo: Statistical and Bayesian learning**

# Statistical Learning Theory

Our starting point is the **underlying law  $y=f(x)$**  which we aim to learn from a dataset  $(x_i, y_i), i=1, \dots, N$ . We will also need an **hypothesis set  $H$**  containing all functions that we consider to be good candidates for the underlying law

The goal of **Statistical Learning Theory** is to determine a function from the hypothesis set  $H$  that approximates  $f(x)$  as best as possible, ideally in a strict mathematical limit

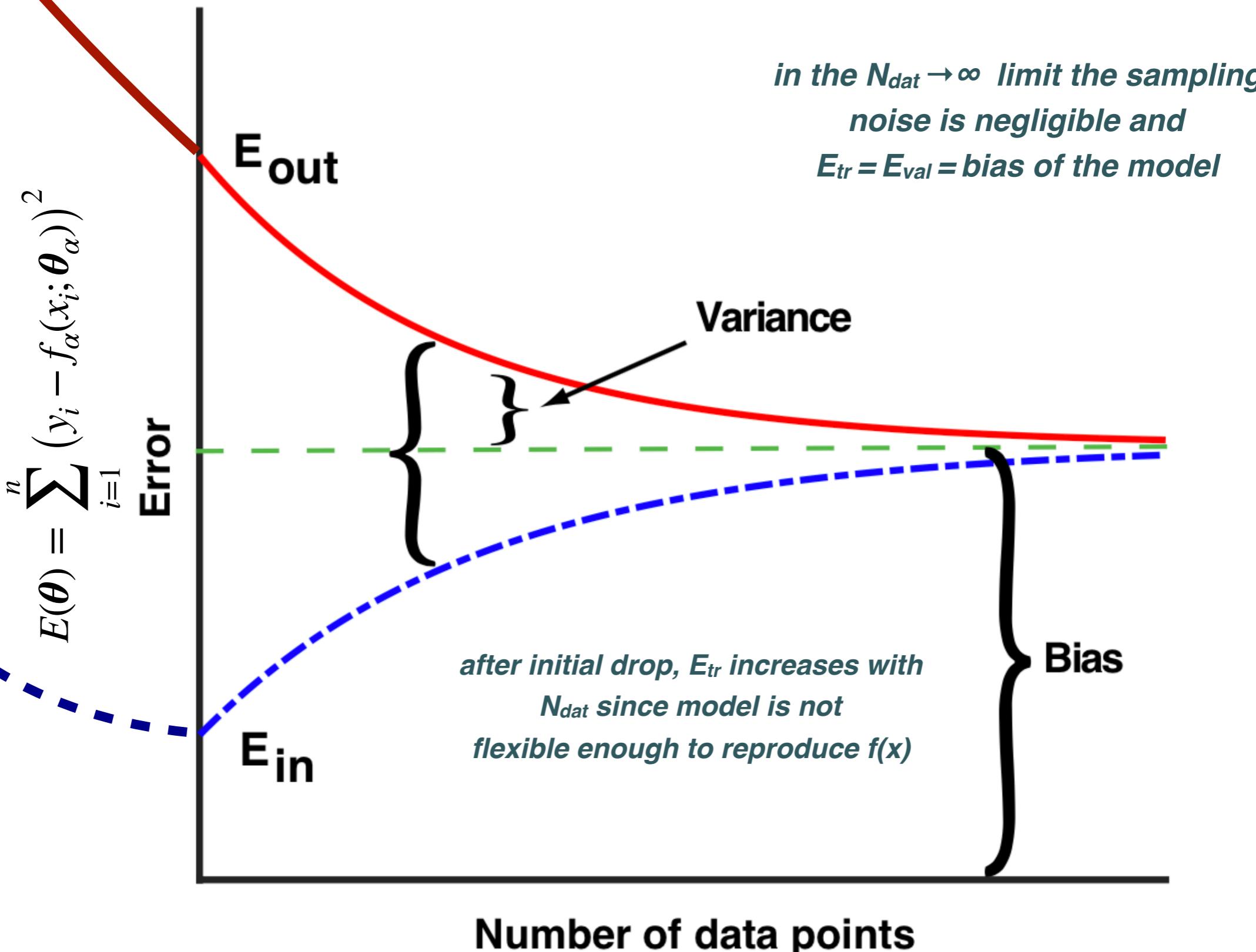
Here we will provide an **intuitive picture** of how Statistical Learning works

Consider the following situation:

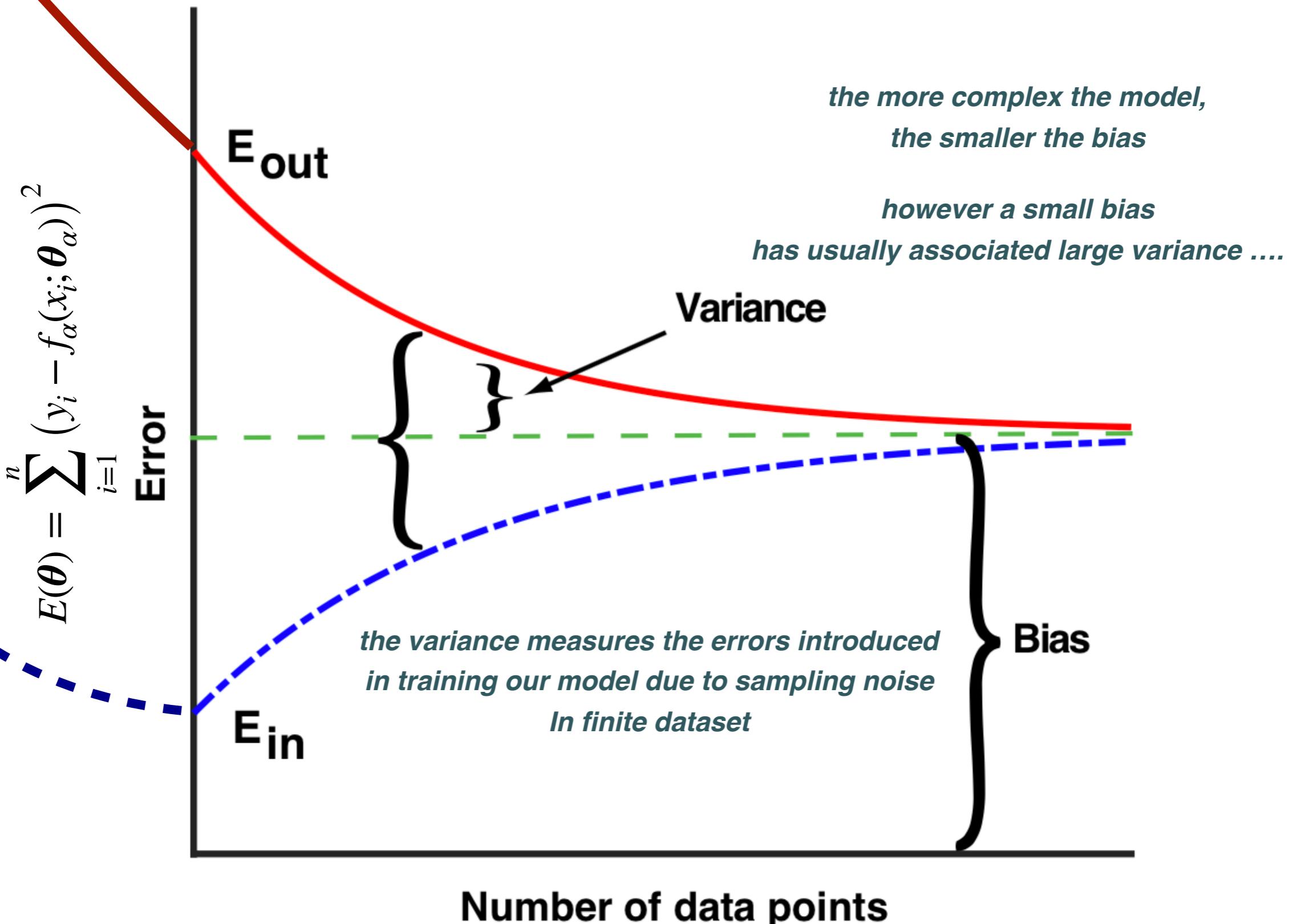
- The underlying law is so complex that we cannot aim to exactly reproduce  $f(x)$
- We want to study the dependence of  $E_{tr}$  and  $E_{val}$  with the number of data points

This setting allows us to present one of the most important concepts in the theory of Machine Learning: **the bias-variance tradeoff**

# Statistical Learning Theory

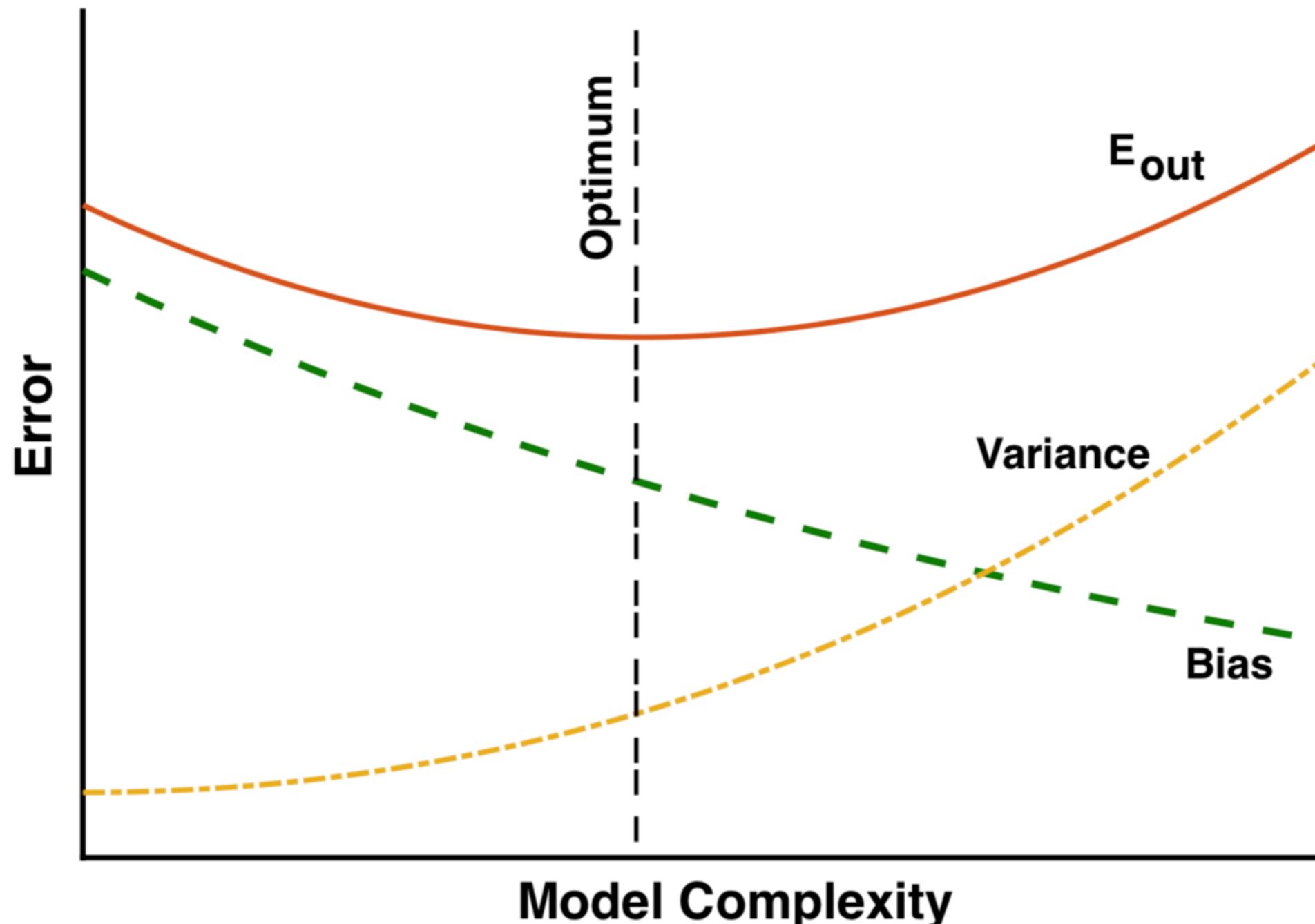


# Statistical Learning Theory



# The bias-variance tradeoff

Optimal model performance (measured by minimising the generalisation error  $E_{\text{val}}$ ) is typically achieved at intermediate levels of model complexity: the **bias-variance tradeoff**

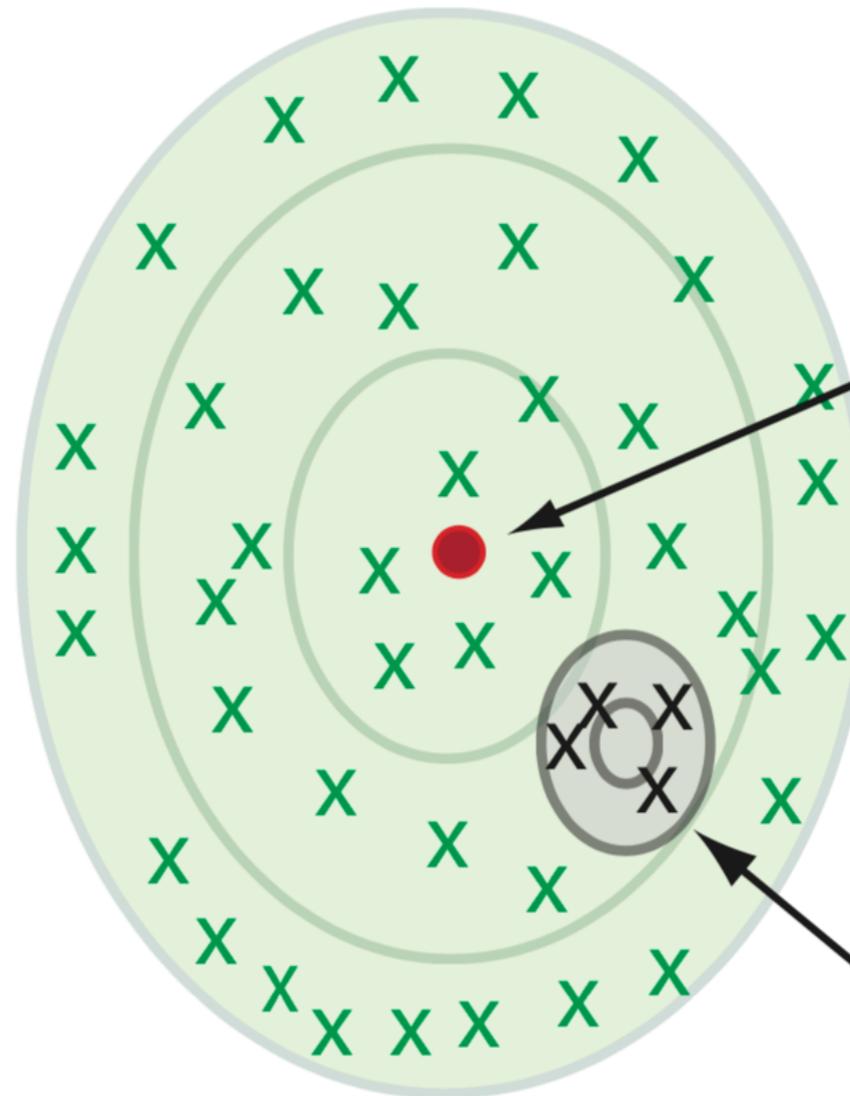


# The bias-variance tradeoff

Optimal model performance (measured by minimising the generalisation error  $E_{\text{val}}$ ) is typically achieved at intermediate levels of model complexity: the **bias-variance tradeoff**

*more complex model*

**High variance,  
low-bias model**



**True model**

*simpler model*

**Low variance,  
high-bias model**

# The bias-variance tradeoff

we can provide further insight on the bias-variance tradeoff with a simple scenario

assume a **model** extracted from a **training set**  $\longrightarrow \hat{y}(x_{\text{tr}})$

and that the true underlying law is given by

$$Y = y(X) + \epsilon, \quad y(x_{\text{tr}}) = E(Y, X = x_{\text{tr}})$$

  
*zero-mean, fixed variance noise*

now take a data point **outside the training set** and compute its variance in our model

$$E[(y_0 - \hat{y}(x_0))^2] = (\text{Bias}[\hat{y}(x_0)])^2 + \text{Var}[\hat{y}(x_0))] + \text{Var}(\epsilon)$$

$$\text{Bias}[\hat{y}(x_0))] = E[\hat{y}(x_0)] - y_0$$

$$\text{Var}[\hat{y}(x_0))] = E[\hat{y}^2(x_0)] - E[\hat{y}(x_0)]^2$$

the **model generalisation power** decreases both with its bias and its variance

# Bayesian Inference

Bayesian inference a method of statistical inference in which **Bayes' theorem** is used to **update the probability for an hypothesis** as more information becomes available

in Bayesian statistics there are two main ingredients:

*likelihood function*

$$p(X | \theta)$$

probability of observing the dataset  $X$  given model parameters  $\theta$

*prior distribution*

$$p(\theta)$$



knowledge of model parameters before we see the data  $X$

which are used to compute the **posterior distribution** using **Bayes' Theorem**

$$p(\theta | X) = \frac{p(X | \theta)p(\theta)}{\int d\theta' p(X | \theta')p(\theta')}$$

probability of the model parameters  $\theta$  after observing the dataset  $X$

# Bayesian Inference

Many common statistical features such as least-squares fitting can be understood as carrying out a **Maximum Likelihood Estimation (MLE)**

In MLE one selects model parameters that **maximise likelihood** of observed data

$$\hat{\theta} = \arg \max_{\theta} \log p(X | \theta)$$

A central ingredient of Bayesian Inference is related to the **choice of prior**

no knowledge on model parameters  $\longrightarrow$  ***uninformative prior***

some knowledge on model parameters  $\longrightarrow$  ***informative prior***

*an informative prior tends to decrease the variance of posterior distribution while potentially, increasing bias*

e.g. **Gaussian prior**  $p(\theta, \lambda) = \prod_j \sqrt{\frac{\lambda}{2\pi}} e^{-\lambda\theta_j^2}$  *justified from belief that many parameters will be small*

# Bayesian Inference

once we have computed using Bayesian Inference the posterior distribution,  
we can evaluate the **optimal values** of the model parameters

posterior mean  $\longrightarrow \langle \theta \rangle = \int d\theta \theta p(X | \theta)$

posterior mode  $\longrightarrow \hat{\theta} = \arg \max_{\theta} p(X | \theta)$

If the prior depends on **hyperparameters such as  $\lambda$**  then one needs also to determine a suitable range for them (a hierarchical prior)

Bayesian methods are very powerful in Machine Learning: Bayesian networks, multinomial & Gaussian naive Bayes....

# **Supervised Learning: Optimisation Strategies**

# Optimisation strategies

problems in **Supervised Machine Learning** are defined by the following ingredients:

**(1) Input dataset:**  $\mathcal{D} = (\mathbf{X}, \mathbf{Y})$

**(2) Model:**  $f(\mathbf{X}; \theta)$

**(3) Cost function:**  $C(\mathbf{Y}; f(\mathbf{X}; \theta))$

the model parameters are then found by **minimising the cost function**

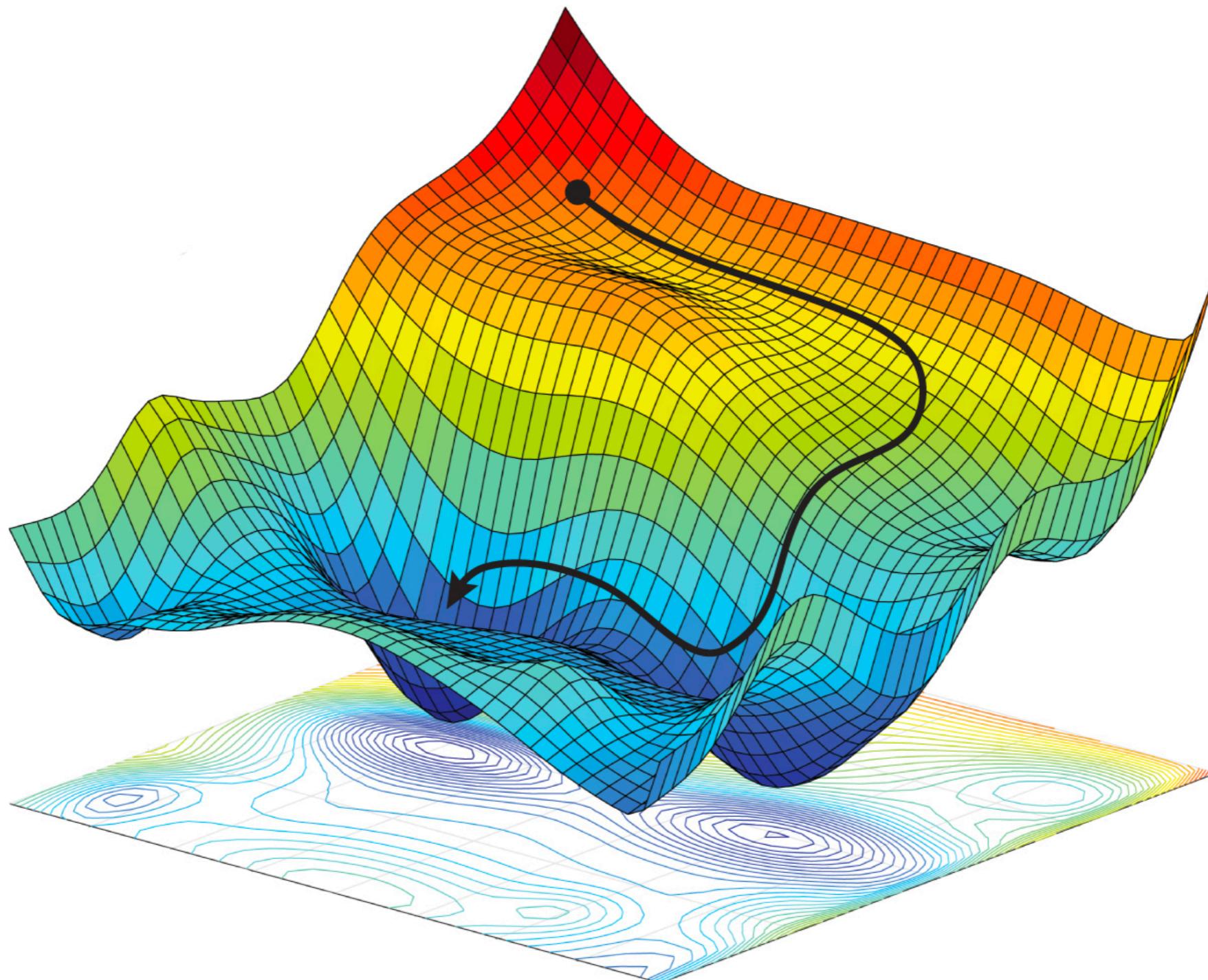
for simple models, such as in polynomial regression, the solution of this minimisation problem can be found analytically. However this is in general not possible with complex models in realistic applications. In the context of Machine Learning studies, there exist two main classes of **minimisers (optimisers)** that are frequently deployed:

- ➊ Gradient Descent and its generalisations

- ➋ Genetic Algorithms and its generalisations

# Gradient descent

basic idea: iteratively adjust the model parameters in the direction where the **gradient of the cost function** is large and negative



# Gradient descent

basic idea: iteratively adjust the model parameters in the direction where the **gradient of the cost function** is large and negative (**steepest descent direction**)

Our goal is thus to **minimise an error (cost) function** that can usually be expressed as

$$E(\theta) = \sum_{i=1}^n e_i(x_i; \theta)$$

e.g. in **linear regression** we had that

$$E(\theta) = \sum_{i=1}^n (y_i - f_\alpha(x_i; \theta_\alpha))^2 \quad \text{so} \quad e_i = (y_i - f_\alpha(x_i; \theta_\alpha))^2$$

and we will see that in **logistic regression** (classification problems)

$$E(\theta) = \sum_{i=1}^n (-y_i \ln \sigma(\mathbf{x}_i^T \theta) - (1 - y_i) \ln [1 - \sigma(\mathbf{x}_i^T \theta)])$$

*aka the cross-entropy, relevant for categorisation*

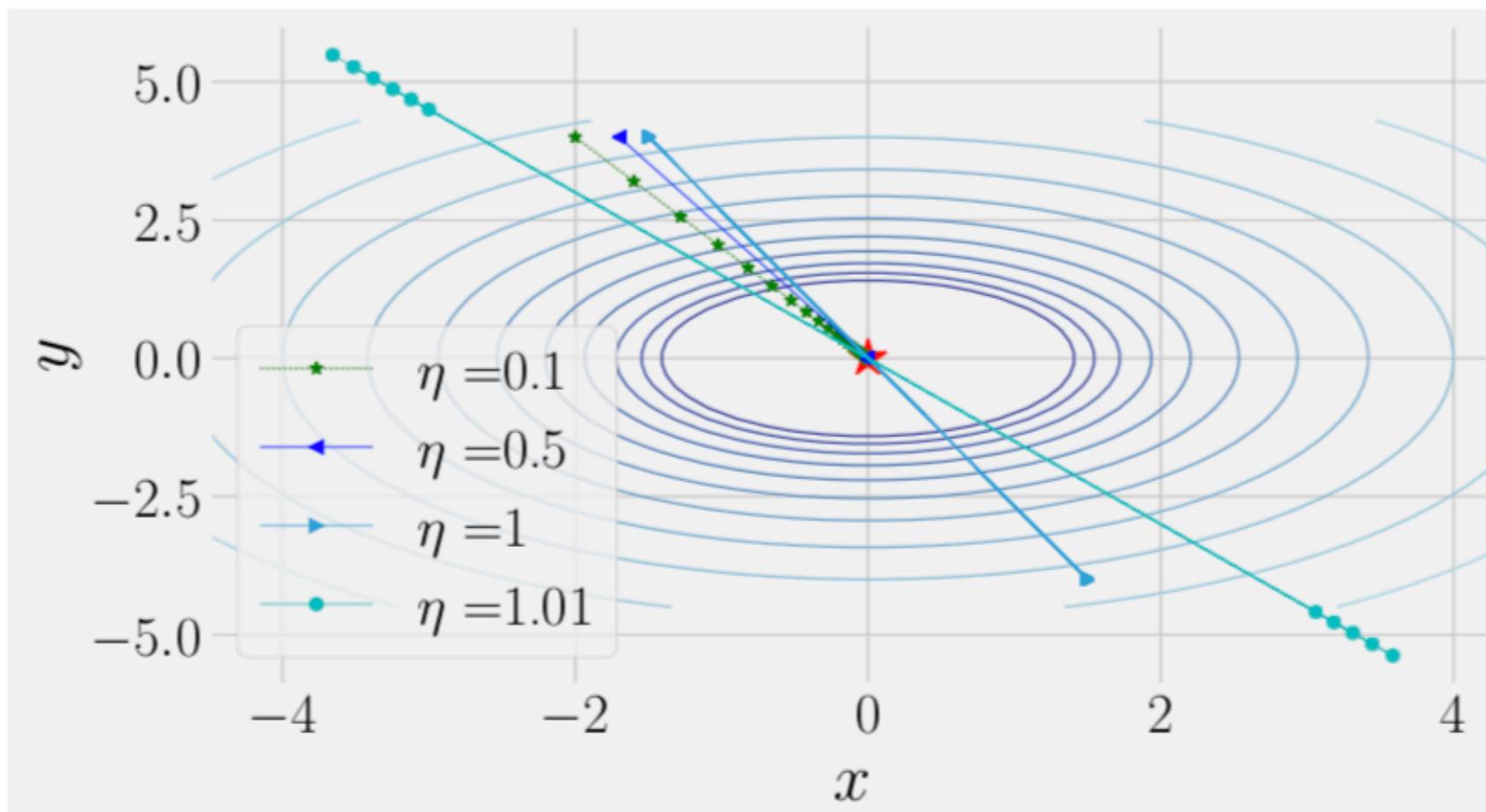
# Gradient descent

starting from a suitable initial condition, GD **iteratively updates** the model parameters:

$$\mathbf{v}_t = \eta_t \nabla_{\theta} E(\theta_t), \quad \theta_{t+1} = \theta_t - \mathbf{v}_t$$

*learning rate*      *gradient of cost function*      *update of model parameters between iterations  $t$  and  $t+1$*

the learning rate determines the size of the step in the direction of the gradient



*small  $\eta$ : guaranteed to find local minimum, at price of large number of iterations*

*large  $\eta$ : can overshoot minimum, problems of convergence*

*nb evaluating gradient can be very cpu-intensive!*

# Gradient descent

starting from a suitable initial condition, GD **iteratively updates** the model parameters:

$$\mathbf{v}_t = \eta_t \nabla_{\theta} E(\boldsymbol{\theta}_t), \quad \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \mathbf{v}_t$$

*learning rate*      *gradient of cost function*      *update of model parameters between iterations  $t$  and  $t+1$*

compare GD with popular **Newton's method**: first Taylor-expand of cost function

$$E(\boldsymbol{\theta} + \mathbf{v}) \simeq E(\boldsymbol{\theta}) + \nabla_{\theta} E(\boldsymbol{\theta}) \mathbf{v} + \frac{1}{2} \mathbf{v}^T H(\boldsymbol{\theta}) \mathbf{v}$$

$$H_{ij}(\boldsymbol{\theta}) = \left\{ \frac{\partial^2 E(\boldsymbol{\theta})}{\partial \theta_i \partial \theta_j} \right\}$$

**Hessian matrix (2nd derivatives)**

# Gradient descent

starting from a suitable initial condition, GD **iteratively updates** the model parameters:

$$\mathbf{v}_t = \eta_t \nabla_{\theta} E(\boldsymbol{\theta}_t), \quad \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \mathbf{v}_t$$

*learning rate*      *gradient of cost function*      *update of model parameters between iterations  $t$  and  $t+1$*

compare GD with popular **Newton's method**: first Taylor-expand of cost function

$$E(\boldsymbol{\theta} + \mathbf{v}) \simeq E(\boldsymbol{\theta}) + \nabla_{\theta} E(\boldsymbol{\theta}) \mathbf{v} + \frac{1}{2} \mathbf{v}^T H(\boldsymbol{\theta}) \mathbf{v}$$

and then differentiate with respect to the step requiring that the **linear term vanishes**

$$\left. \frac{\partial E(\boldsymbol{\theta} + \mathbf{v})}{\partial \mathbf{v}} \right|_{\mathbf{v}_{\text{opt}}} = 0 \longrightarrow \nabla_{\theta} E(\boldsymbol{\theta}) + H(\boldsymbol{\theta}) \mathbf{v}_{\text{opt}} = 0$$

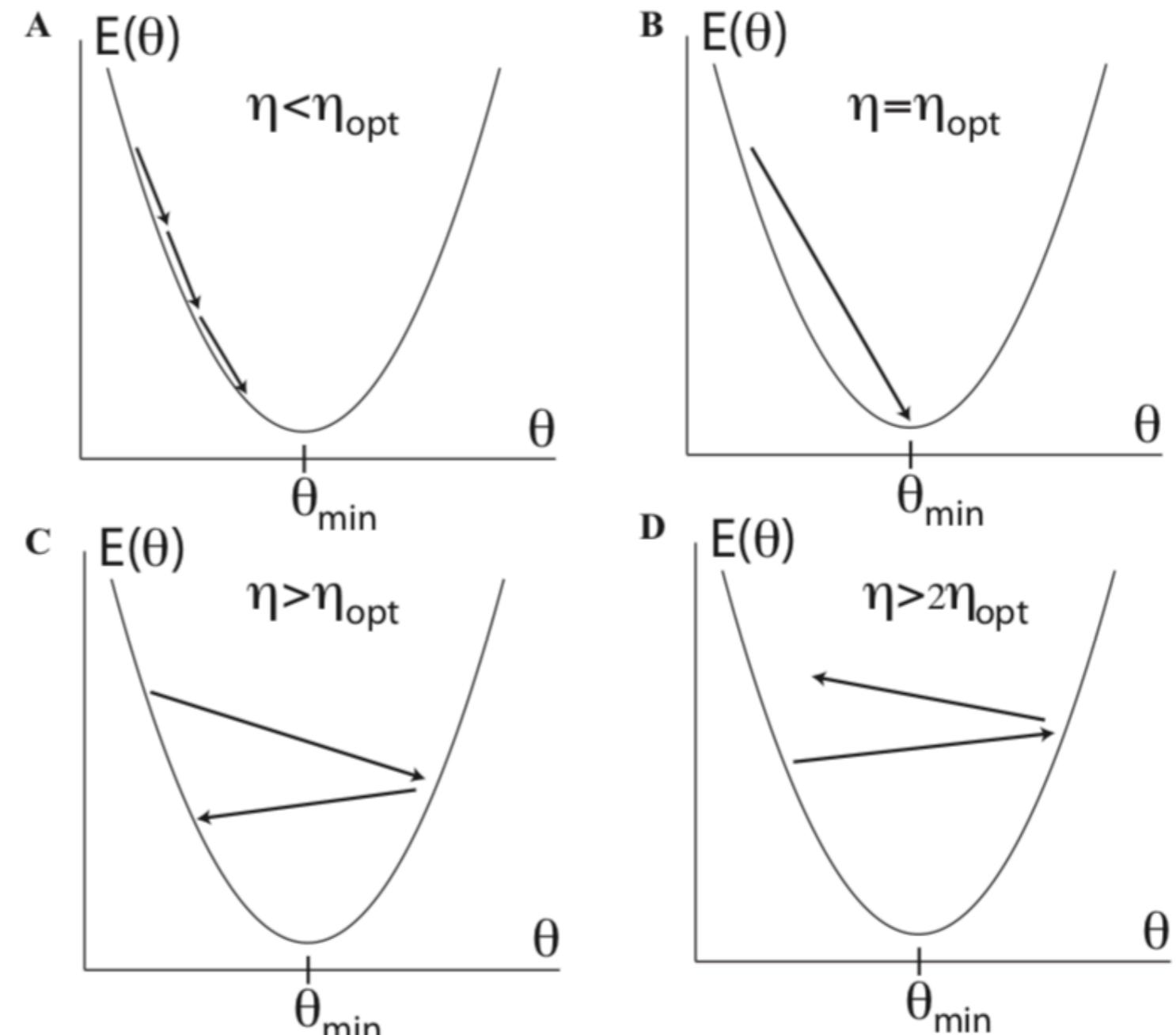
$$\mathbf{v}_t = H^{-1}(\boldsymbol{\theta}_t) \cdot \nabla_{\theta} E(\boldsymbol{\theta}_t), \quad \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \mathbf{v}_t$$

# Gradient descent

Gradient Descent  $\longrightarrow \mathbf{v}_t = \eta_t \nabla_{\theta} E(\theta_t), \quad \theta_{t+1} = \theta_t - \mathbf{v}_t$

Newton's method  $\longrightarrow \mathbf{v}_t = H^{-1}(\theta_t) \cdot \nabla_{\theta} E(\theta_t), \quad \theta_{t+1} = \theta_t - \mathbf{v}_t$

- Newton's method not practical for ML applications: it involves **evaluation and inversion of Hessian matrices with  $N^2$  entries**, with  $N$  = number of model parameters



- But it provides useful ideas about how to improve GD, for example by **adapting the learning rate to the local curvature** of region of the parameter space

# Advanced gradient descent

the simplest implementation of GD, unsurprisingly, has several limitations

- It converges to **local, rather than global**, minima of the cost function

*poor performance for realistic applications*

# Advanced gradient descent

the simplest implementation of GD, unsurprisingly, has several limitations

- It converges to **local, rather than global**, minima of the cost function

*poor performance for realistic applications*

- Evaluating gradients is **computationally expensive** for large datasets

$$E(\boldsymbol{\theta}) = \sum_{i=1}^n (y_i - f_\alpha(x_i; \boldsymbol{\theta}_\alpha))^2$$

*involves sum over all n data points*

# Advanced gradient descent

the simplest implementation of GD, unsurprisingly, has several limitations

- It converges to **local, rather than global**, minima of the cost function

*poor performance for realistic applications*

- Evaluating gradients is **computationally expensive** for large datasets

$$E(\theta) = \sum_{i=1}^n (y_i - f_\alpha(x_i; \theta_\alpha))^2$$

*involves sum over all n data points*

- Very sensitive to choice of **learning rates**

*Ideally one would like an adaptive learning rate*

# Advanced gradient descent

the simplest implementation of GD, unsurprisingly, has several limitations

- It converges to **local, rather than global**, minima of the cost function

*poor performance for realistic applications*

- Evaluating gradients is **computationally expensive** for large datasets

$$E(\boldsymbol{\theta}) = \sum_{i=1}^n (y_i - f_\alpha(x_i; \boldsymbol{\theta}_\alpha))^2$$

*involves sum over all n data points*

- Very sensitive to choice of **learning rates**

*Ideally one would like an adaptive learning rate*

- Treats **uniformly all directions** in the parameter space

*we would like to use info also on curvature (as in Newton's method)*

**Generalised Gradient Descent methods** have been developed to address these shortcomings and are at the basis of **modern deep learning methods**

# Stochastic gradient descent

Stochasticity can be added to GD by approximating the gradient on a subset of the training data, called a **mini-batch**

$$\nabla_{\theta} E(\theta) = \sum_{i=1}^n \nabla_{\theta} e_i(x_i; \theta) \rightarrow \nabla_{\theta} E^{\text{MB}}(\theta) \equiv \sum_{i \in B_k} \nabla_{\theta} e_i(x_i; \theta)$$

$k = 1, \dots, n/M \leftarrow \# \text{ points per batches}$

↑  
 $\# \text{ batches}$

We then **cycle over all mini-batches**, updating the model parameters at each step  $k$

$$\mathbf{v}_t = \eta_t \nabla_{\theta}^{\text{MB}} E(\theta_t), \quad \theta_{t+1} = \theta_t - \mathbf{v}_t$$

A full iteration over all  $n$  data points (over the  $n/M$  batches) is called an **epoch**

benefits of SGD: stochasticity prevents getting stuck in local minima, the calculation of the gradient is speed up & stochasticity adds as natural regulariser

# Adding momentum to SGD

SGD can be used with a **momentum term** that provides some memory on the direction in which one is moving in the parameter space

$$\mathbf{v}_t = \gamma \mathbf{v}_{t-1} + \eta_t \nabla_{\theta}^{\text{MB}} E(\theta_t), \quad \theta_{t+1} = \theta_t - \mathbf{v}_t$$



*momentum parameter ( $0 < \gamma < 1$ )*

$$\Delta \theta_{t+1} = \gamma \Delta \theta_t - \eta_t \nabla_{\theta} E(\theta_t), \quad \Delta \theta_t = \theta_t - \theta_{t-1}$$

momentum in SGD helps to **gain speed in directions with persistent but small gradients**, while suppressing oscillations in high-curvature directions.

# Adaptive Gradient Descent

**Adaptive GD** vary the learning rate to reflect the **local curvature** of the parameter space without the need to evaluate the Hessian matrix

**RMSprop**: keep track also of the **second moment of gradient**, similar as how the momentum term is a running average of previous gradients

$$\theta_{t+1} = \theta_t - \eta_t \frac{\mathbf{g}_t}{\sqrt{s_t + \epsilon}}$$

$$\mathbf{g}_t = \nabla_{\theta} E(\theta) \quad s_t = \beta s_{t-1} + (1 - \beta) \mathbf{g}_t^2$$

# Adaptive Gradient Descent

**Adaptive GD** vary the learning rate to reflect the **local curvature** of the parameter space without the need to evaluate the Hessian matrix

**ADAM:** adaptively change the parameters from info on running averages of first and second moments of the gradient

$$\mathbf{g}_t = \nabla_{\theta} E(\theta)$$

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} - (1 - \beta_1) \mathbf{g}_t \quad \mathbf{s}_t = \beta_2 \mathbf{s}_{t-1} + (1 - \beta_2) \mathbf{g}_t^2$$

$$\widehat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{1 - (\beta_1)^t}$$

*sets lifetime  
of first moment*

$$\widehat{\mathbf{s}}_t = \frac{\mathbf{s}_t}{1 - (\beta_2)^t}$$

*sets lifetime  
of second moment*

$$\theta_{t+1} = \theta_t - \eta_t \frac{\widehat{\mathbf{m}}_t}{\sqrt{\widehat{\mathbf{s}}_t + \epsilon}}$$

# Adaptive Gradient Descent

**Adaptive GD** vary the learning rate to reflect the **local curvature** of the parameter space without the need to evaluate the Hessian matrix

**ADAM** has two main advantages: (i) adapting step size to cut off large gradient directions (prevent oscillations) and (ii) measuring gradients in a natural length scale

to see this, express the ADAM update rules in terms of the **variance in parameter space**

$$\sigma_t = \hat{s}_t - (\hat{m}_t)^2$$

and we can see that the update rule for one of the parameters reads now

$$\Delta\theta_{t+1} = -\eta_t \frac{\hat{m}_t}{\sqrt{\sigma_t^2 + \hat{m}_t^2} + \epsilon}$$

*small fluctuations  
of gradient*

$$\Delta\theta_{t+1} \rightarrow -\eta_t$$

*large fluctuations  
of gradient*

$$\Delta\theta_{t+1} = -\eta_t \hat{m}_t / \sigma_t^2$$

*learning rate promotional to signal-to-noise ratio*

# Exercise II

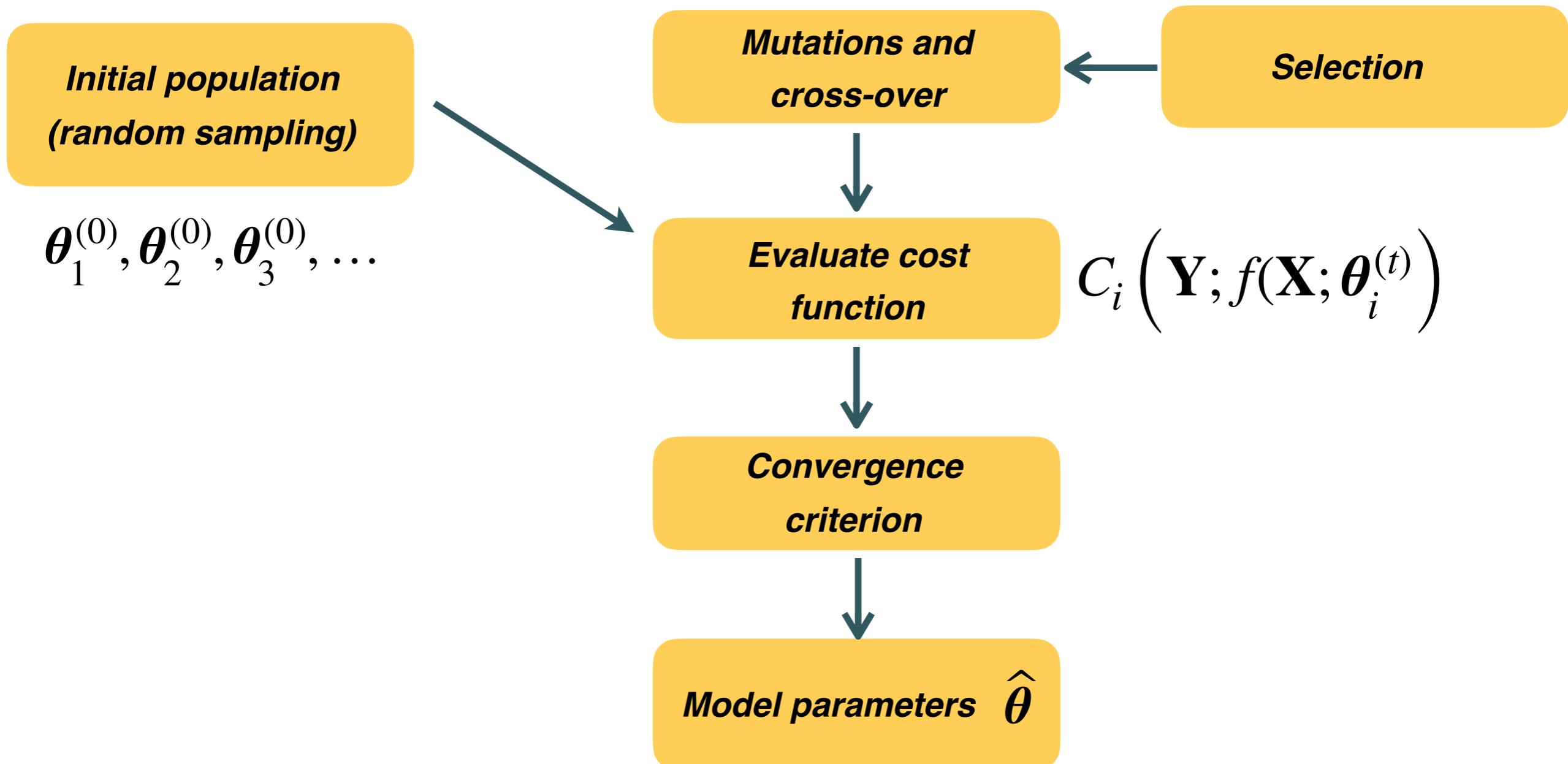
starting point is the **Python script** that you will find in

<https://github.com/juanrojochacon/ml-ditp-attp/blob/master/Tutorials/Tutorial1/Lecture1-GradientDescent.py>

- 📌 Study different optimisation algorithms based on GD
- 🎧 Determine which choices of the parameters affect more the results

# Genetic Algorithms

GAs combine stochastic and deterministic ingredients to explore the model parameter space and minimise the cost function



# Genetic Algorithms

# Genetic Algorithms

One can improve the efficiency of GA incorporating global (in addition to local) information

One example is the Covariance matrix Adaptation - Evolutionary Strategy (CMA-ES)

Start by randomly initialising the model parameters using a Gaussian distribution

$$\boldsymbol{\theta}^{(0)} \sim \mathcal{N}(0, \mathbf{C}^{(0)})$$

At every generation of the GA, we generate N mutants as follows

$$\mathbf{a}_k^{(t)} \sim \boldsymbol{\theta}^{(t-1)} + \sigma^{(t-1)} \mathcal{N}(0, \mathbf{C}^{(t-1)}) , \quad k = 1, \dots, N$$

Then the new search centre is computed as weighted average over a fixed fraction of best mutants

$$\boldsymbol{\theta}_k^{(t)} \sim \boldsymbol{\theta}^{(t-1)} + \sum_{k=1} W_k (\mathbf{a}_k - \boldsymbol{\theta})$$

# Genetic Algorithms

