Machine Learning for Physics and Astronomy

Juan Rojo

VU Amsterdam & Theory group, Nikhef

Natuur- en Sterrenkunde BSc (Joint Degree), Honours Track

Lecture 2, 14/09/2021

Lecture outline

- Course logistics: Recap
- Why machine learning is difficult: Recap
- Deterministic optimisation algorithms: gradient descent
- Stochastic optimisation algorithms: genetic algorithms

Logistics Recap

Course schedule

Week	Monday	Tuesday	Wednesday	Thursday
36 6/9 10/9		11-13 HC: Lecture dr. Juan Rojo Chacon SP G2.13	9-11 LC: Practical session dr. Juan Rojo Chacon SP B0.202	
37 13/9 17/9		11-13 HC: Lecture dr. Juan Rojo Chacon SP G2.13	9-11 LC: Practical session dr. Juan Rojo Chacon SP B0.202	
38 20/9 24/9		11-13 HC: Lecture dr. Juan Rojo Chacon SP G2.13	9-11 LC: Practical session dr. Juan Rojo Chacon SP B0.202	
39 27/9 1/10		11-13 HC: Lecture dr. Juan Rojo Chacon SP G2.13	9-11 LC: Practical session dr. Juan Rojo Chacon SP B0.202	
40 4/10 8/10		11-13 HC: Lecture dr. Juan Rojo Chacon SP G2.13	9-11 LC: Practical session dr. Juan Rojo Chacon SP B0.202	
41 11/10 15/10		11-13 HC: Lecture dr. Juan Rojo Chacon SP G2.13	9-11 LC: Practical session dr. Juan Rojo Chacon SP B0.202	
42 18/10 22/10		11-13 HC: Lecture dr. Juan Rojo Chacon SP G2.13	9-11 LC: Practical session dr. Juan Rojo Chacon SP B0.202	

- **7 lectures** (Tuesdays 11am) and **7 tutorials** (Thursdays 9am)
- All course activities will be **on campus**. Recordings of the zoom lectures of the **2020-2021 academic year** are available on Canvas.
- Additional discussion sessions can be scheduled upon request

Course logistics

- All the **course material** (slides, recordings, notebooks for the tutorials, other resources) will be made available via the **Canvas page of the course**, which will also be used for the **main announcements** as well as for the submission of **homework assignments**

- This material will also be available via the course **GitHub repository,** which will always contain the most updated versions of the tutorial's codes

https://github.com/LHCfitNikhef/ML4PA

note that this repo can be directly linked to e.g. **Google's Collab** to run the codes on the cloud rather than locally

- Furthermore, we have created a **Discord server** to streamline the communications between the instructors and the students, and to facilitate discussions during the lectures. You should all have received an invitation to join the course channel, else get in touch

Course evaluation

- Students should write a short report (around 6 pages) about a specific **application of Machine Learning algorithms** to a physics or astronomy problem that they find interesting. This report and the subsequent presentation count up to **60%** of the course grade.

You need to submit this report via Canvas by **Monday 1st November**

the recorded presentations of the 2020/2021 cohort are available on Canvas for inspiration

- In addition, after each tutorial we will propose some short **homework assignments** that need to be completed and submitted via Canvas. These homework assignments count up to **40%** of the final course grade

Homework assignments must be handed in **within one week** following the tutorial (see Canvas for specific deadlines)

Tutorials

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

- The **Python notebooks** with the course tutorials can be either run locally (if you have an updated python installation) or on the cloud by uploading the notebook in **Google Collab**
- Please note that some tutorials require the download of heavy datasets, make sure this is done before the tutorial
- Due to time constraints we can only offer limited assistance with **software installation** - please make sure you can install and execute the notebooks beforehand, and get back to us via Discord if there is any trouble
- The tutorial sessions will be coordinated by Tommaso Giani and Ryan van Mastrigt, the course TAs. Please contact then for any **questions related to the course tutorials**

The tutorials will include both **basic, intermediate,** and **advanced** topics concerning ML algorithms - it is up to you to decide how far you want to push!

Guest lectures 2020/2021

In the previous year we had several guest lecturers presenting state-of-the-art ML applications for physics and astronomy problems within our community

- Lecture 4: **Dr. Atul Chhotray** (applications to Astronomy)
- Lecture 5: **Dr. Christoph Weniger** (applications to Dark Matter)
- Lecture 6: **Dr. Sascha Caron** (applications for High-Energy Physics)
- Lecture 7: **Dr. Tristan Bereau** (applications to Condensed Matter)

Slides and recordings of these guest lectures are also **available on Canvas:** students are encouraged to watch them as extra study resources

Lecture 1 Recap

Supervised learning

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

(1) Input dataset: $\mathscr{D} = (\boldsymbol{X}, \boldsymbol{Y})$

*array of **independent** variables*

*array of **dependent** variables*

$$\boldsymbol{X} = \left(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\right) \qquad \boldsymbol{Y} = \left(y_1, y_2, \ldots, y_N\right)$$

$$\boldsymbol{x}_i = \left(x_{i,1}, x_{i,2}, \ldots, x_{i,p}\right)$$

each independent variable contains p features

Supervised learning

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

(1) Input dataset: $\mathscr{D} = (\boldsymbol{X}, \boldsymbol{Y})$

*array of **independent** variables*

*array of **dependent** variables*

$$\boldsymbol{X} = \left(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\right) \qquad \boldsymbol{Y} = \left(y_1, y_2, \ldots, y_n\right)$$

$$\boldsymbol{x}_i = \left(x_{i,1}, x_{i,2}, \ldots, x_{i,p}\right)$$

each independent variable contains p features

(2) Model: $f\left(\boldsymbol{X}, \boldsymbol{\theta}\right)$

mapping between dependent and independent variables

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

model parameters

$$\boldsymbol{\theta} = \left(\theta_1, \theta_2, \ldots, \theta_m\right)$$

the more complex the problem, the more flexible the model

Supervised learning

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

(1) Input dataset: $\mathscr{D} = (\boldsymbol{X}, \boldsymbol{Y})$

(2) Model: $f\left(\boldsymbol{X}, \boldsymbol{\theta}\right)$

(3) Cost function: $C\left(\boldsymbol{Y}; f(\boldsymbol{X}; \boldsymbol{\theta}\right)$

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

$$C\left(\boldsymbol{Y}; f(\boldsymbol{X}; \boldsymbol{\theta}\right) = \frac{1}{n} \sum \left(y_i - f(\boldsymbol{x}_i, \boldsymbol{\theta})\right)^2$$

Supervised learning

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

(1) Input dataset: $\mathscr{D} = (\boldsymbol{X}, \boldsymbol{Y})$

(2) Model: $f\left(\boldsymbol{X}, \boldsymbol{\theta}\right)$

(3) Cost function: $C\left(\boldsymbol{Y}; f(\boldsymbol{X}; \boldsymbol{\theta}\right)$

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

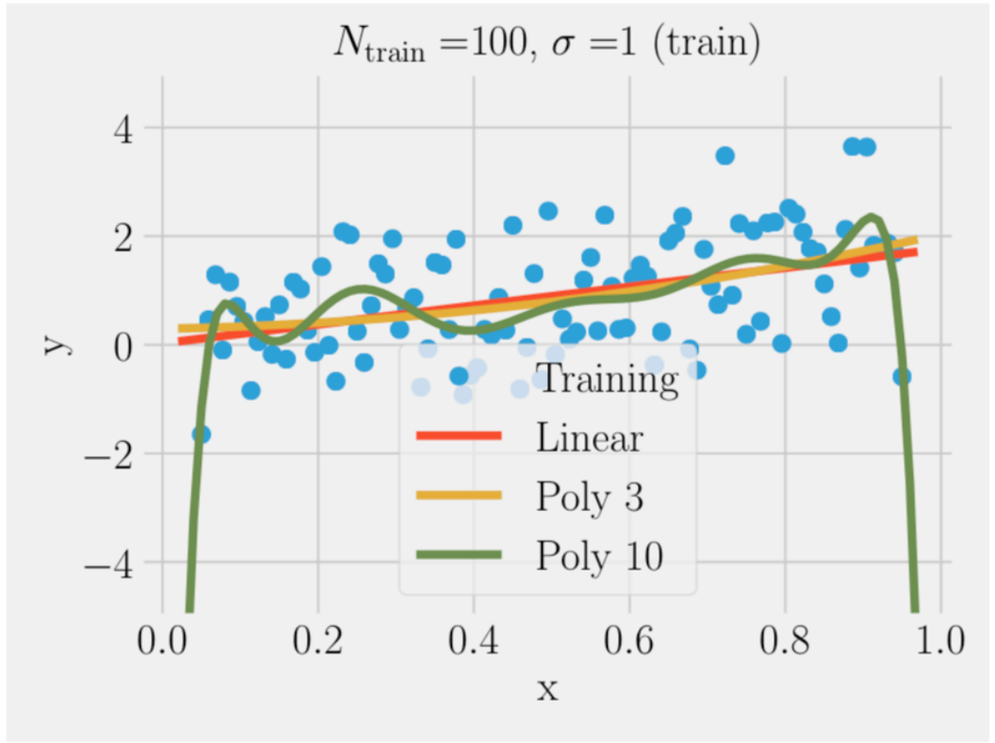
Fitting the model means determining the values of its parameters which **minimise the cost function**

$$\left.\frac{\partial C\left(\boldsymbol{Y}; f(\boldsymbol{X}; \boldsymbol{\theta}\right)}{\partial \theta_i}\right|_{\theta=\theta_{\text{opt}}} = 0$$

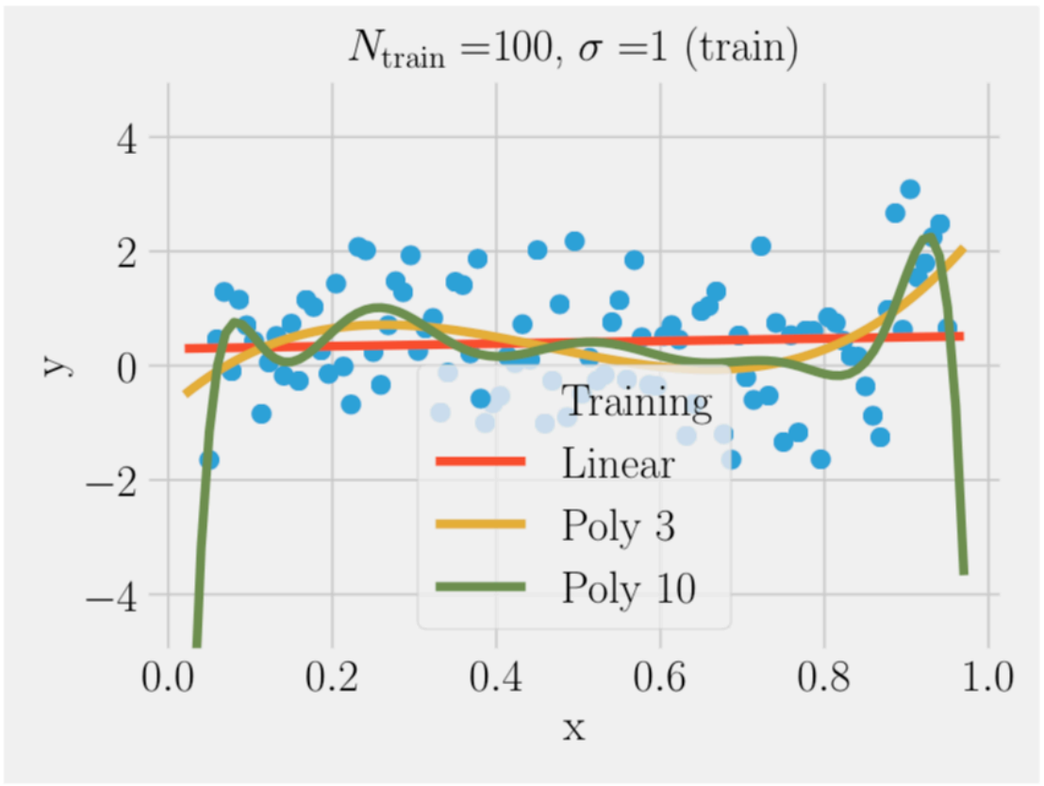
easy right?

Model fitting

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



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



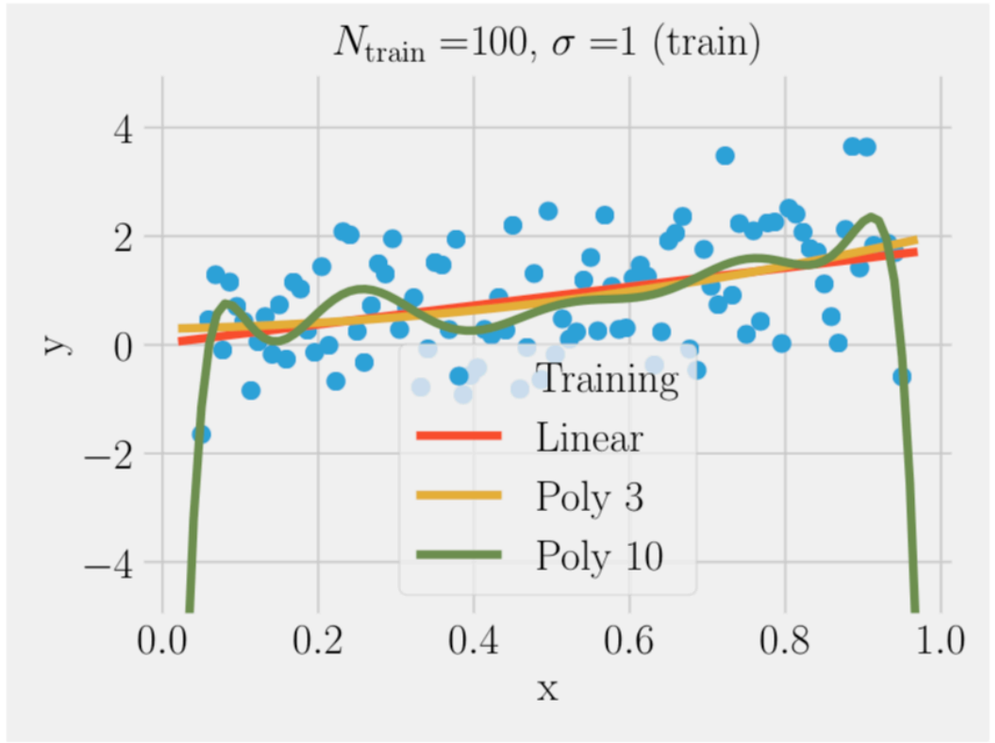
What was the main take-home message of this exercise?

How do you expect these issues could affect realistic machine learning applications

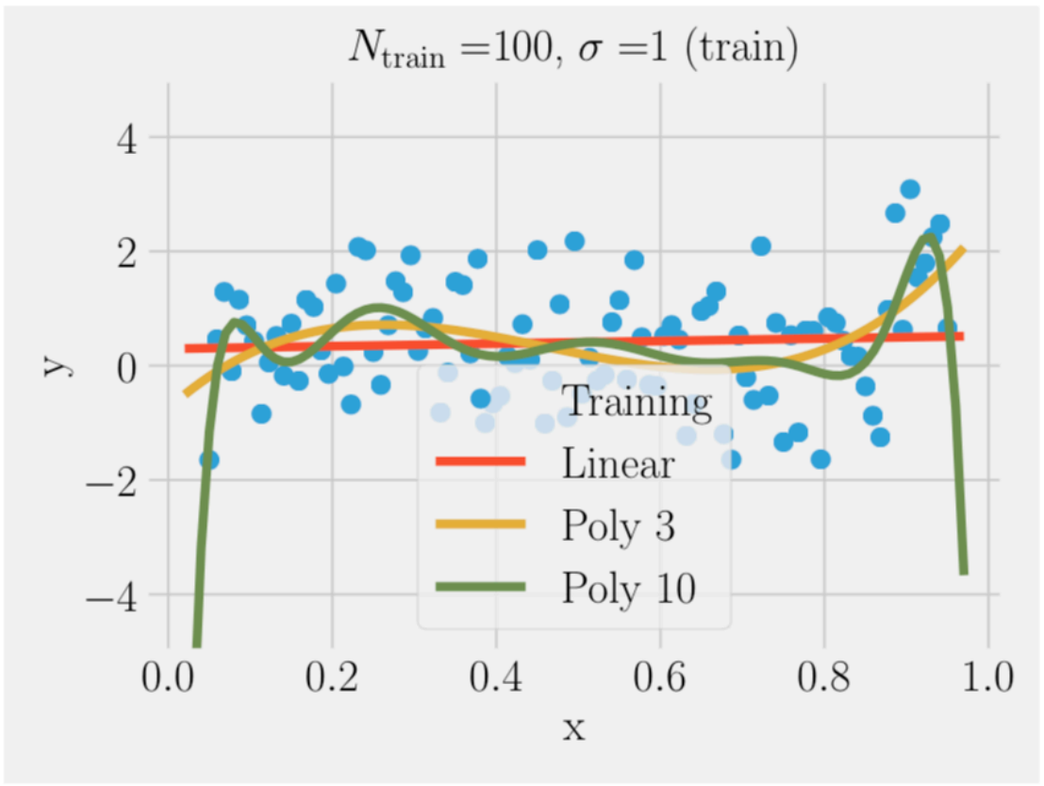
e.g. self-driving cars

Model fitting

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



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



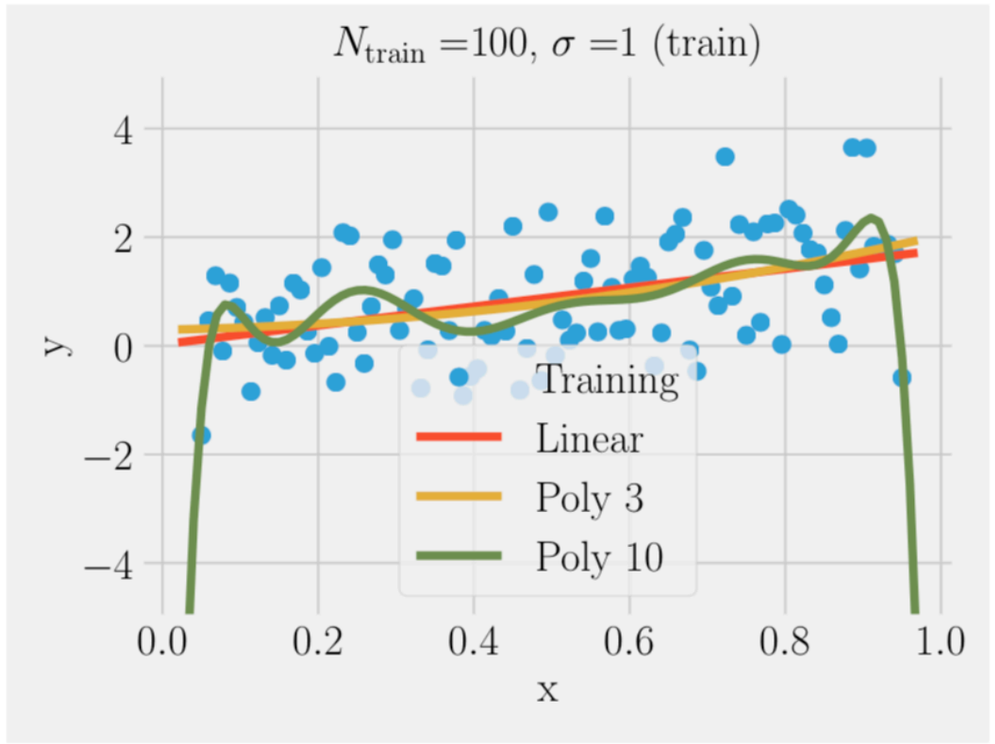
What is the main take-home message of this exercise?

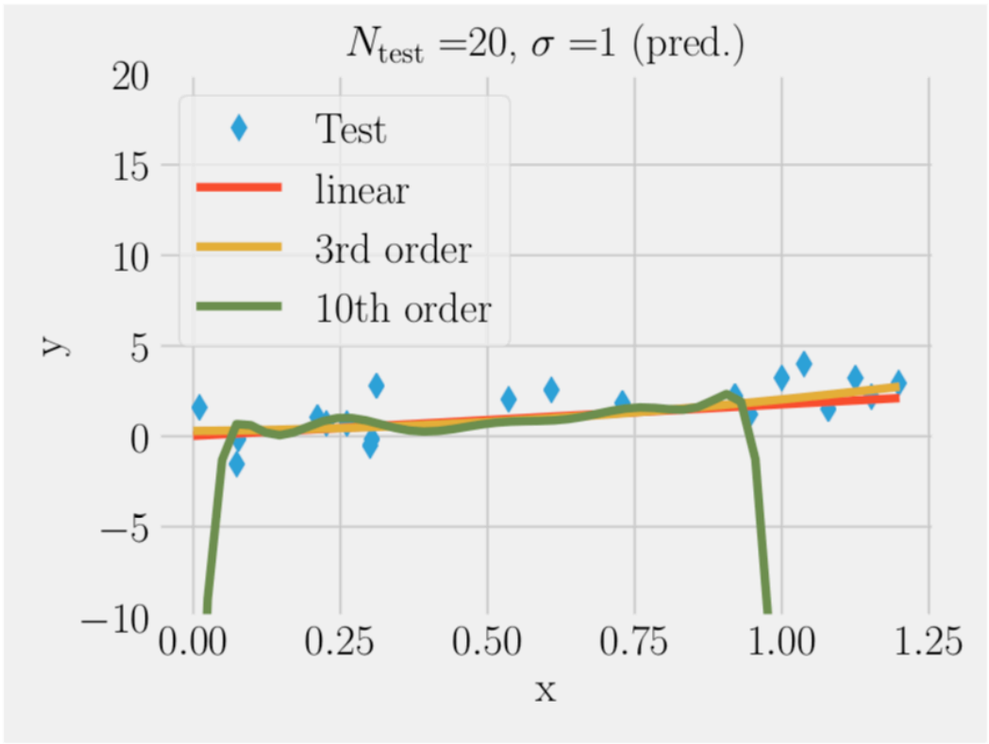
How do you expect these issues could affect realistic machine learning applications

Fitting is not Predicting!

Model fitting

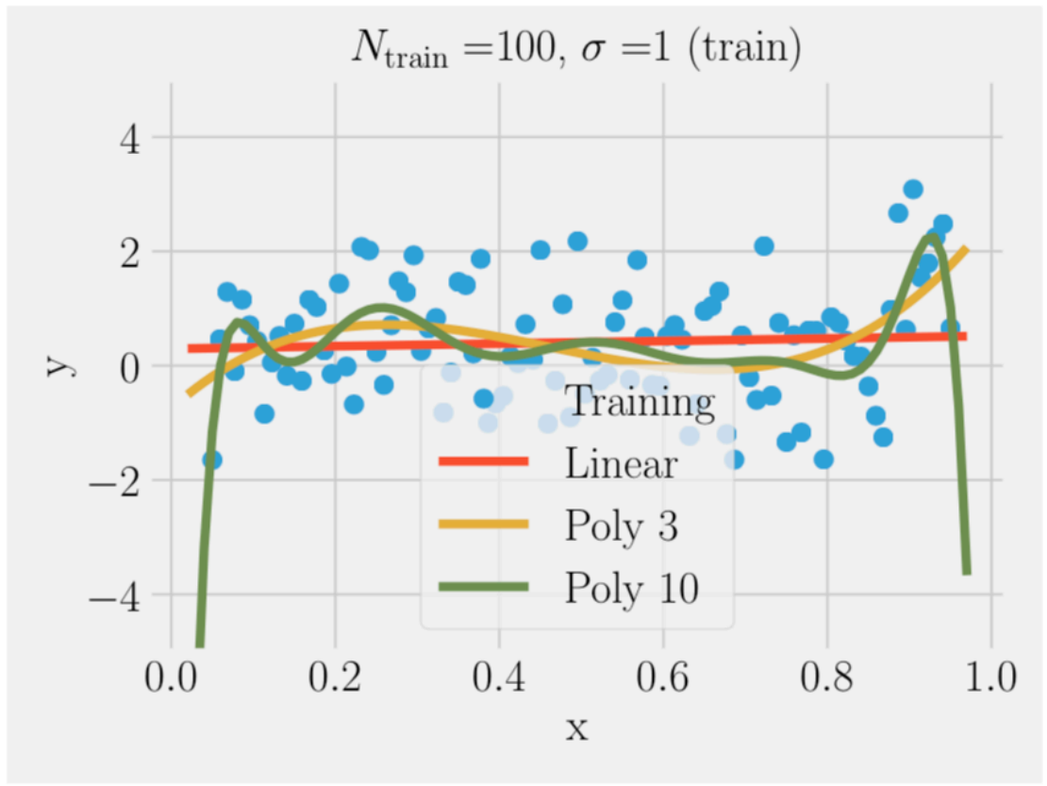
$f(x) = 2x\,, \quad x \in [0,1]$

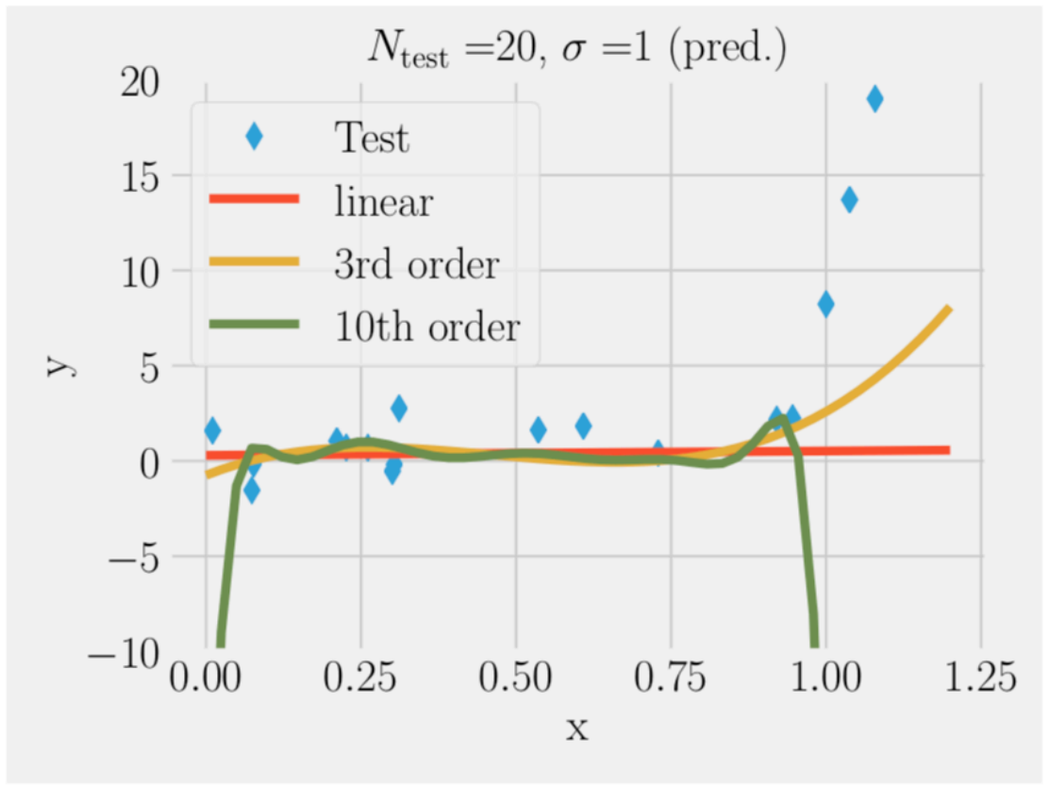




$f(x) = 2x - 10x^5 + 15x^{10}\,, \quad x \in [0,1]$

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





Why Machine Learning 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 ……

Deep Learning successes boosted by hardware developments

Supervised Learning:

Deterministic Optimisation Strategies

Optimisation strategies

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

(1) Input dataset: $\mathscr{D} = (\boldsymbol{X}, \boldsymbol{Y})$

(2) Model: $f\left(\boldsymbol{X}, \boldsymbol{\theta}\right)$

(3) Cost function: $C\left(\boldsymbol{Y}; f(\boldsymbol{X}; \boldsymbol{\theta}\right)$

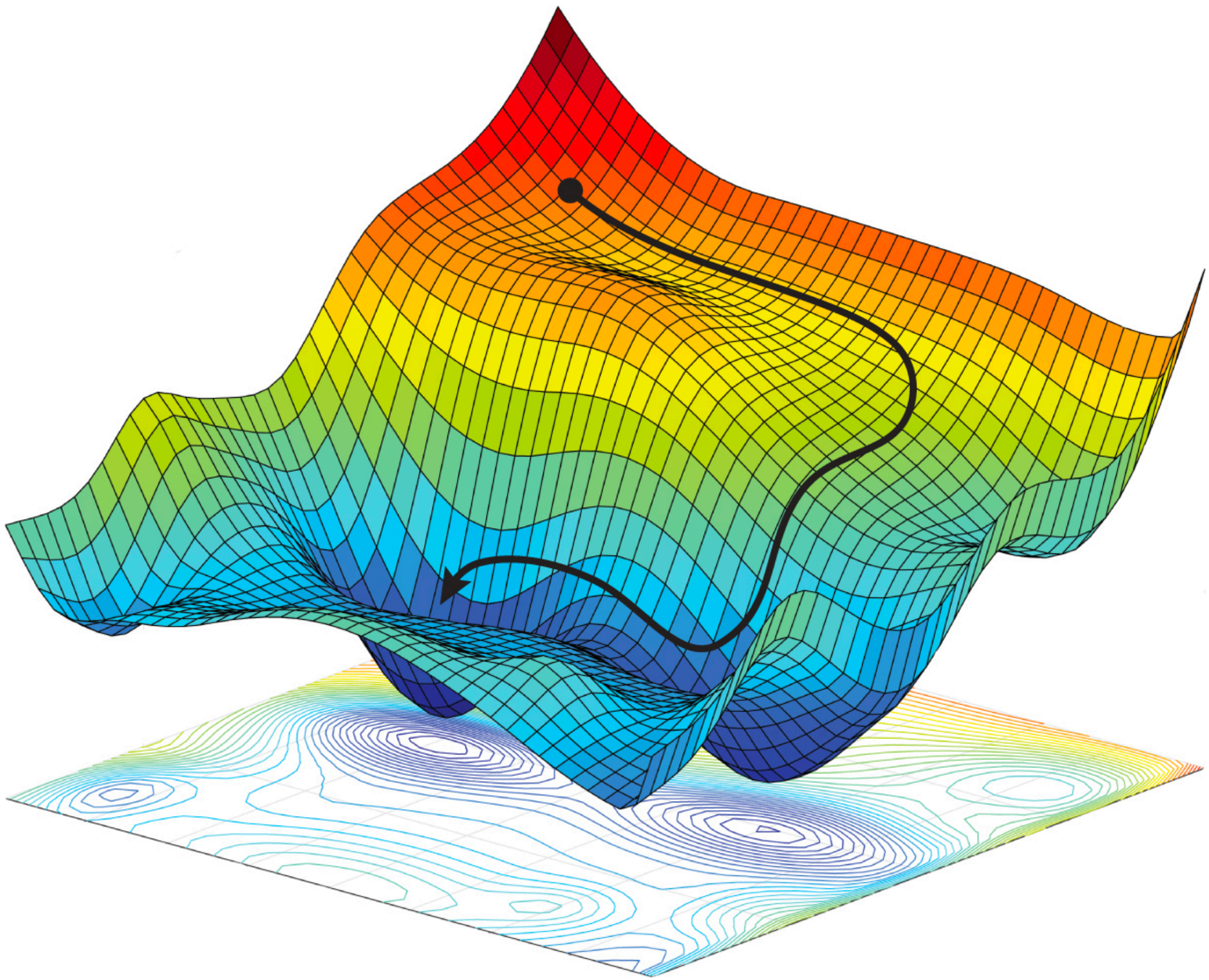
the model parameters are then found by **minimising the cost function**

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

- **Gradient Descent** and its generalisations
- **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 (**steepest descent direction**)



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

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

e.g. in polynomial regression we had that

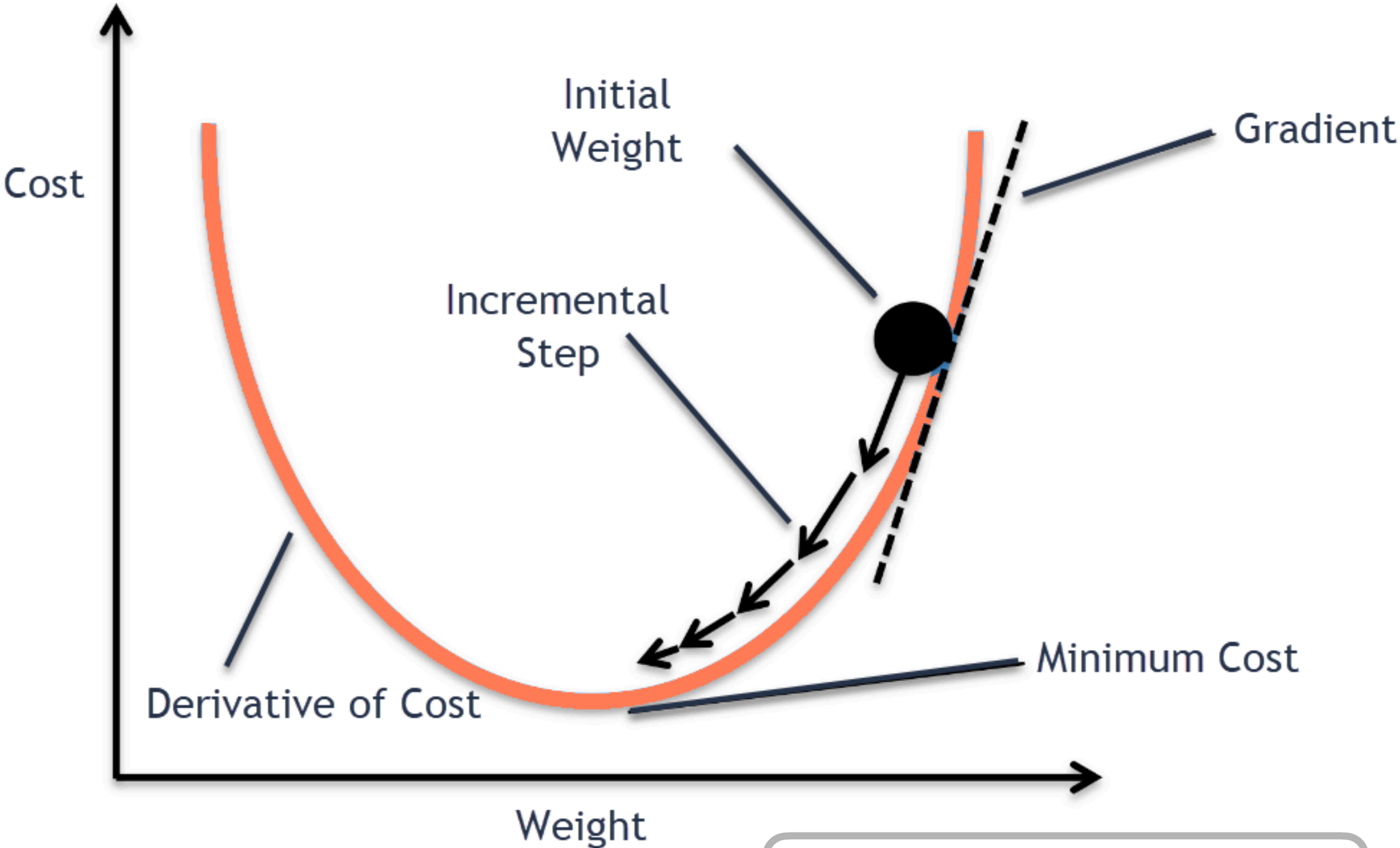
$$E(\boldsymbol{\theta}) = \sum_{i=1}^{n} \left(y_i - f_\alpha(x_i;\boldsymbol{\theta}_\alpha)\right)^2 \quad \text{so} \quad e_i = \left(y_i - f_\alpha(x_i;\boldsymbol{\theta}_\alpha)\right)^2$$

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

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

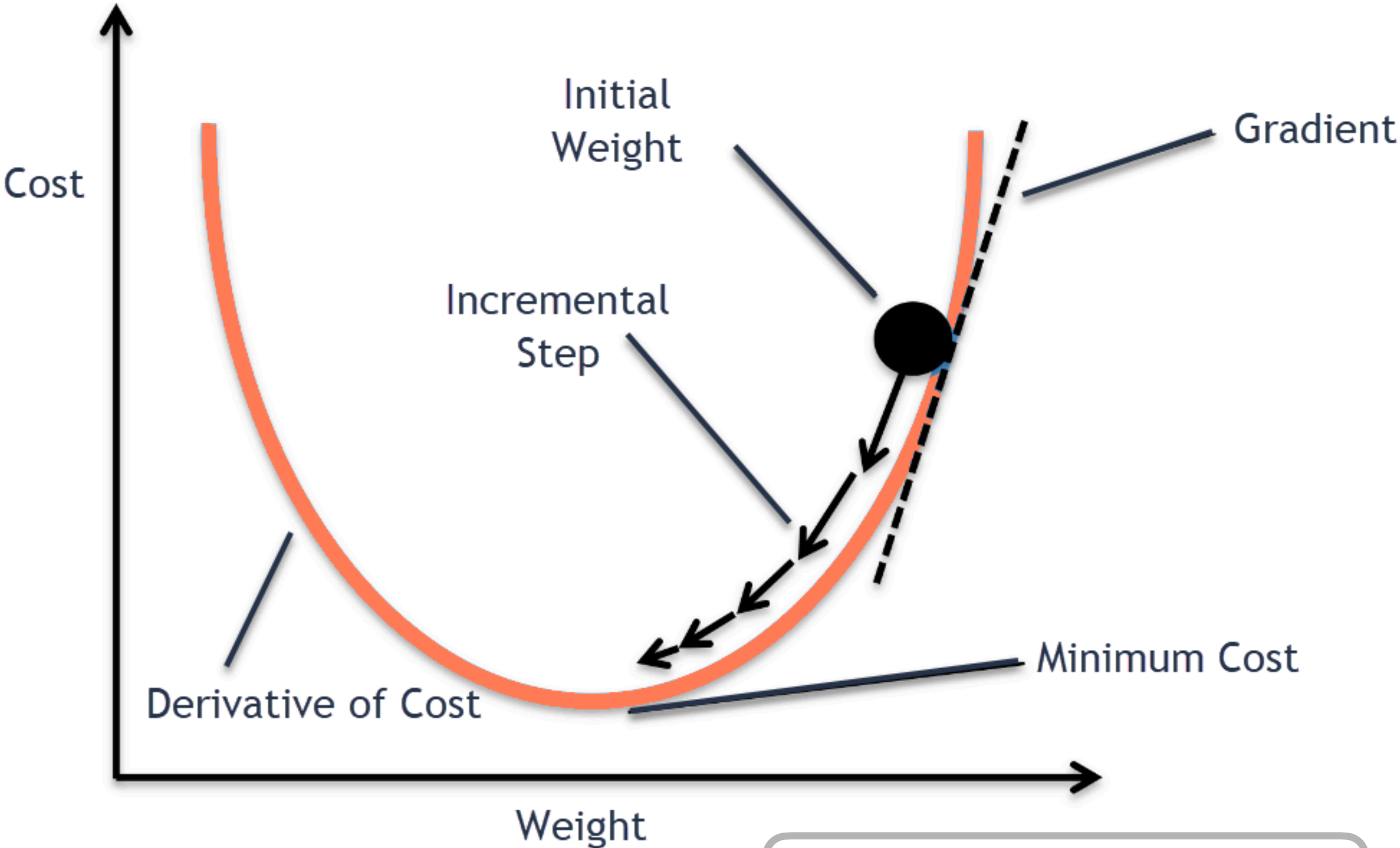
aka the cross-entropy, relevant for categorisation

Gradient descent



What do you think is key benefit of GD?

Gradient descent



Only local information (gradient) required!

Gradient descent

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

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

learning rate

gradient of cost function

*update of model parameters between iterations **t** and **t+1***

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

What could go wrong when choosing the learning rate?

Gradient descent

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

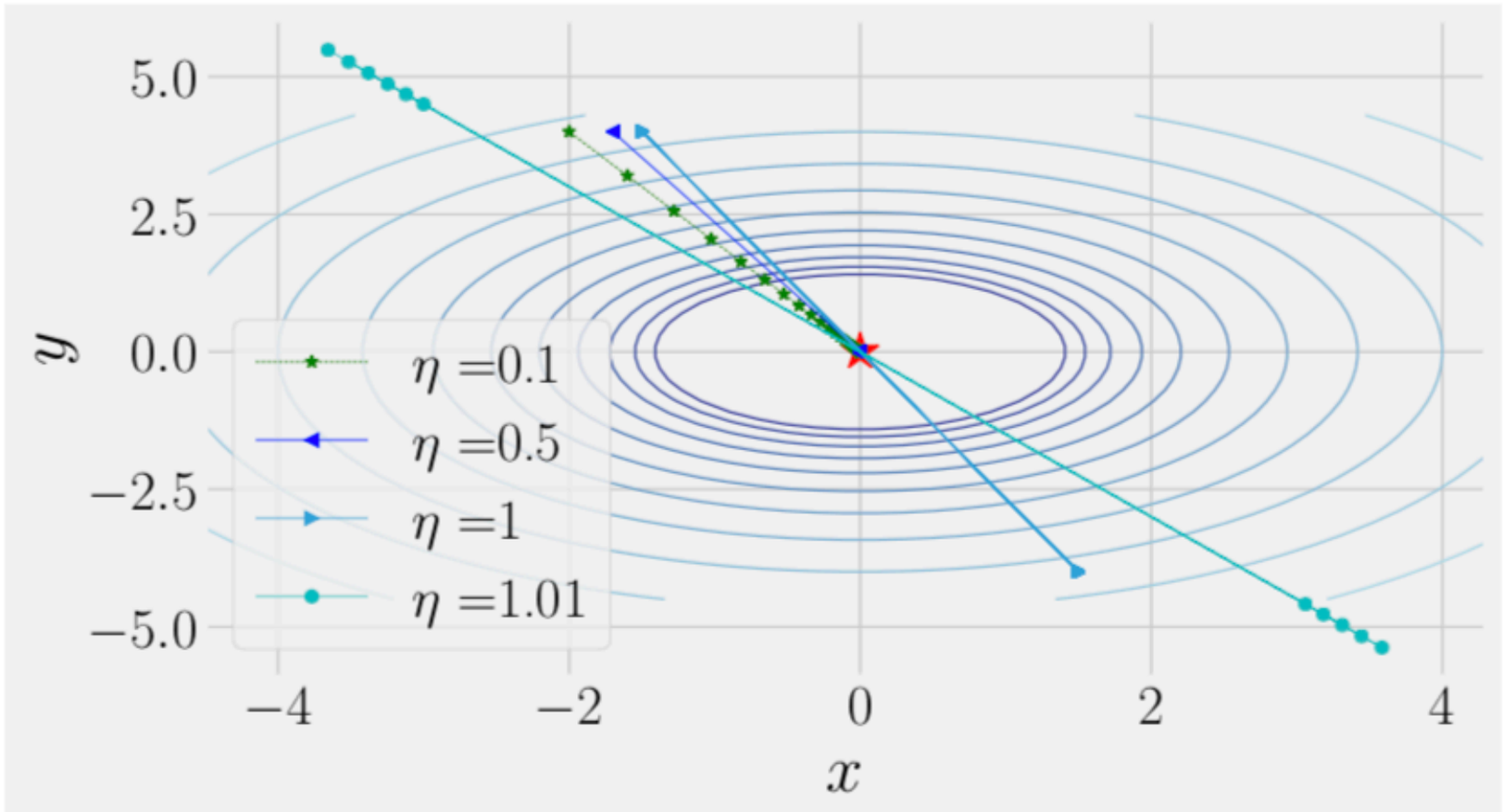
$$\mathbf{v}_t = \eta_t \nabla_\theta E(\boldsymbol{\theta}_t)\,, \qquad \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \mathbf{v}_t$$

learning rate

gradient of cost function

*update of model parameters between iterations **t** and **t+1***

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



small η: guaranteed to find local minimum, at price of large number of iterations

large η: 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:

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

learning rate

gradient of cost function

*update of model parameters between iterations **t** and **t+1***

compare GD with **Newton's method**: first Taylor-expand of cost function for a small change of the model parameters

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

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

Hessian matrix (2nd derivatives)

Gradient descent

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

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

learning rate

gradient of cost function

*update of model parameters between iterations **t** and **t+1***

compare GD with **Newton's method**: first Taylor-expand of cost function for a small change of the model parameters

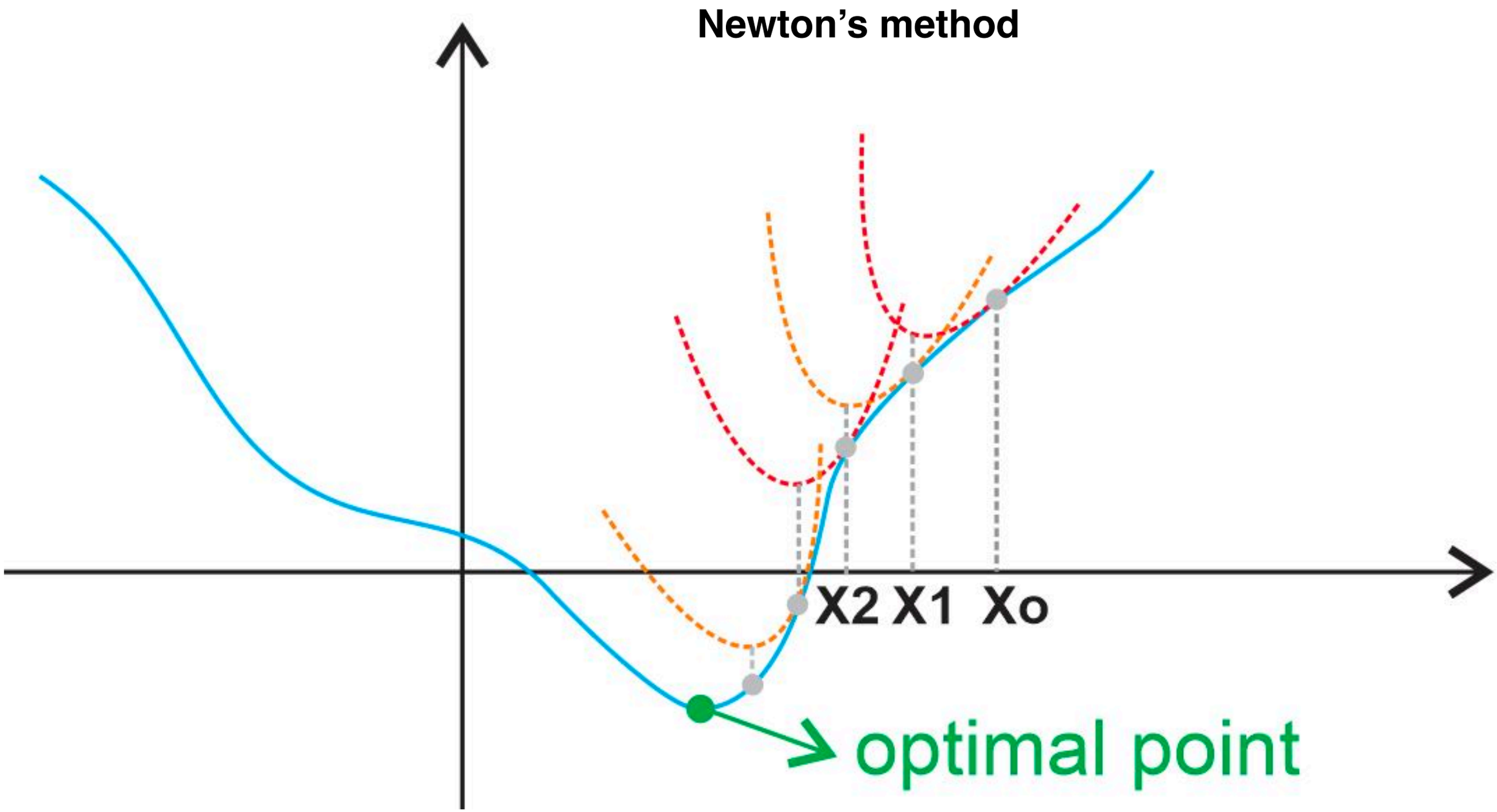
$$E\left(\boldsymbol{\theta} + \mathbf{v}\right) \simeq E\left(\boldsymbol{\theta}\right) + \nabla_\theta E\left(\boldsymbol{\theta}\right) \cdot \mathbf{v} + \frac{1}{2}\mathbf{v}^T H(\boldsymbol{\theta})\mathbf{v}$$

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

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

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

Gradient descent



Gradient descent

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

Newton's method → $\mathbf{v}_t = H^{-1}(\theta_t) \cdot \nabla_\theta E(\boldsymbol{\theta}_t)\,, \qquad \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \mathbf{v}_t$

Why Newton's method not suitable for machine learning problems?

Gradient descent

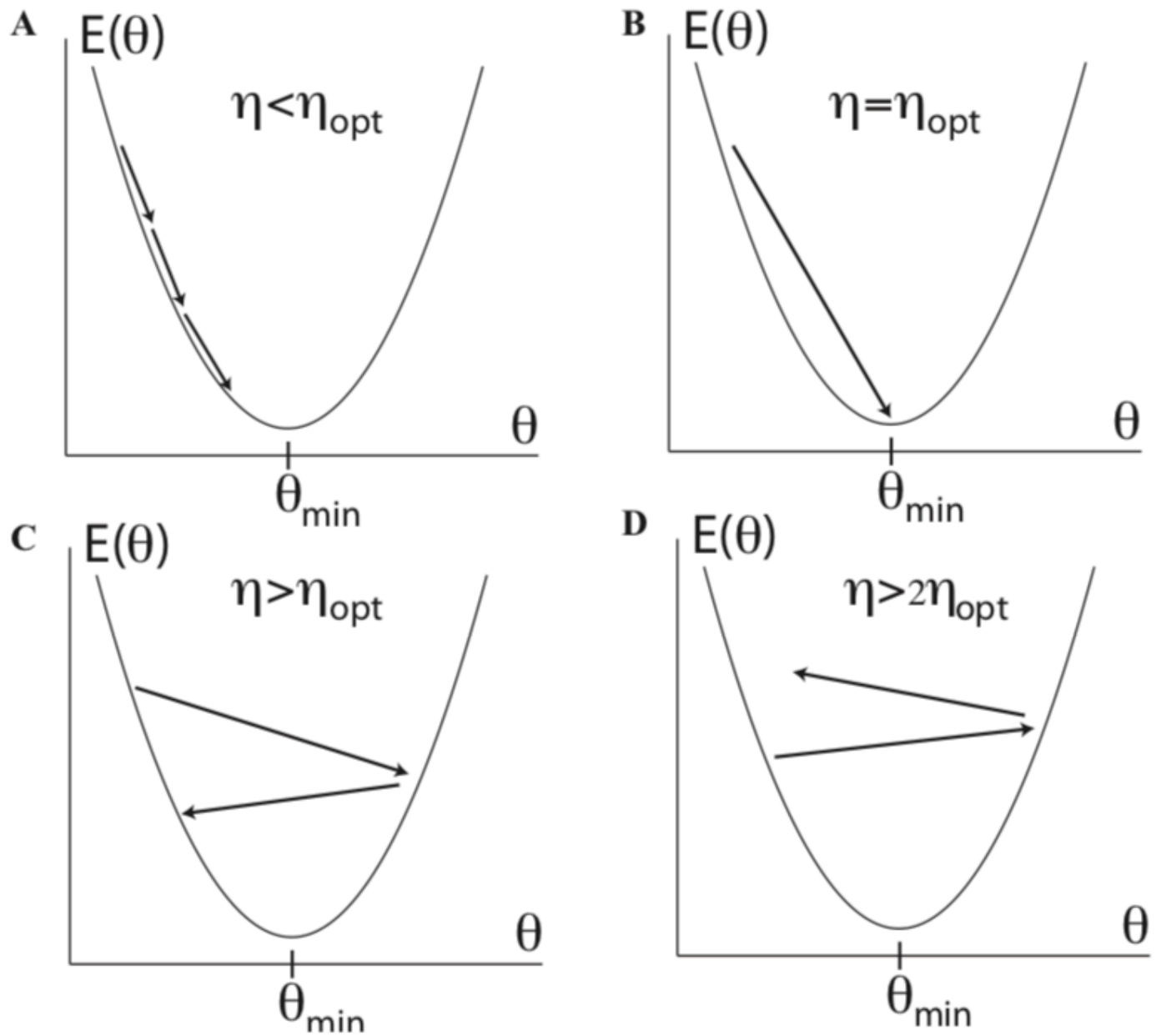
Gradient Descent ⟶ $\mathbf{v}_t = \eta_t \nabla_\theta E(\boldsymbol{\theta}_t)\,, \qquad \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \mathbf{v}_t$

Newton's method ⟶ $\mathbf{v}_t = H^{-1}(\theta_t) \cdot \nabla_\theta E(\boldsymbol{\theta}_t)\,, \qquad \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \mathbf{v}_t$

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

suggest improvements of GD!



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

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

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

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

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

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

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

and why this is a problem??

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

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

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**

$$\nabla_\theta E(\boldsymbol{\theta}) = \sum_{i=1}^{n} \nabla_\theta e_i(\boldsymbol{x}_i;\boldsymbol{\theta}) \simeq \nabla_\theta E^{\mathrm{MB}}(\boldsymbol{\theta}) \equiv \sum_{i\in B_k} \nabla_\theta e_i(\boldsymbol{x}_i;\boldsymbol{\theta})$$

$k = 1,\ldots,n/K$ ← ***# points per batches***

 # batches

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

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

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

benefits of SGD: stochasticity prevents getting stuck in local minima, the calculation of the gradient is speed up & stochasticity acts 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

$$\mathbf{v}_t = \gamma \mathbf{v}_{t-1} + \eta_t \nabla_\theta^{\mathrm{MB}} E(\boldsymbol{\theta}_t)\,, \qquad \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \mathbf{v}_t$$

momentum parameter (0 < ɣ < 1)

$$\Delta\boldsymbol{\theta}_{t+1} = \gamma \Delta\boldsymbol{\theta}_t - \eta_t \nabla_\theta E(\boldsymbol{\theta}_t), \qquad \Delta\boldsymbol{\theta}_t = \boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1}$$

proposed parameter change in iteration t+1

proposed parameter change in iteration t

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

Adaptative Gradient Descent

Adaptative GD varies 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

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \frac{\boldsymbol{g}_t}{\sqrt{\boldsymbol{s}_t + \epsilon}}$$

regulator to avoid numerical divergences

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

gradient at iteration t

$$\boldsymbol{s}_t = \beta \boldsymbol{s}_{t-1} + (1-\beta)\boldsymbol{g}_t^2$$

controls averaging time of 2nd moment of gradient

$$\boldsymbol{s}_t = \mathbb{E}[\boldsymbol{g}_t^2]$$

2nd moment of gradient (averaged over iterations)

$$\beta = 1 \rightarrow \boldsymbol{s}_t = \boldsymbol{s}_{t-1}$$
$$\beta = 0 \rightarrow \boldsymbol{s}_t = \boldsymbol{g}_t^2$$

effective learning rate reduced in directions where gradient is consistently large

Adaptative Gradient Descent

Adaptative GD varies 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 information on **running averages** of first and second moments of the gradient

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

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

$$\widehat{\boldsymbol{m}}_t = \frac{\boldsymbol{m}_t}{1-(\beta_1)^t} \qquad \widehat{\boldsymbol{s}}_t = \frac{\boldsymbol{s}_t}{1-(\beta_2)^t}$$

sets lifetime of first moment

sets lifetime of second moment

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \frac{\widehat{\boldsymbol{m}}_t}{\sqrt{\widehat{\boldsymbol{s}}_t + \epsilon}}$$

Adaptative Gradient Descent

Adaptative GD varies 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**

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

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

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

large fluctuations of gradient

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

learning rate promotional to signal-to-noise ratio

Tutorial 2 (2a & 2b)

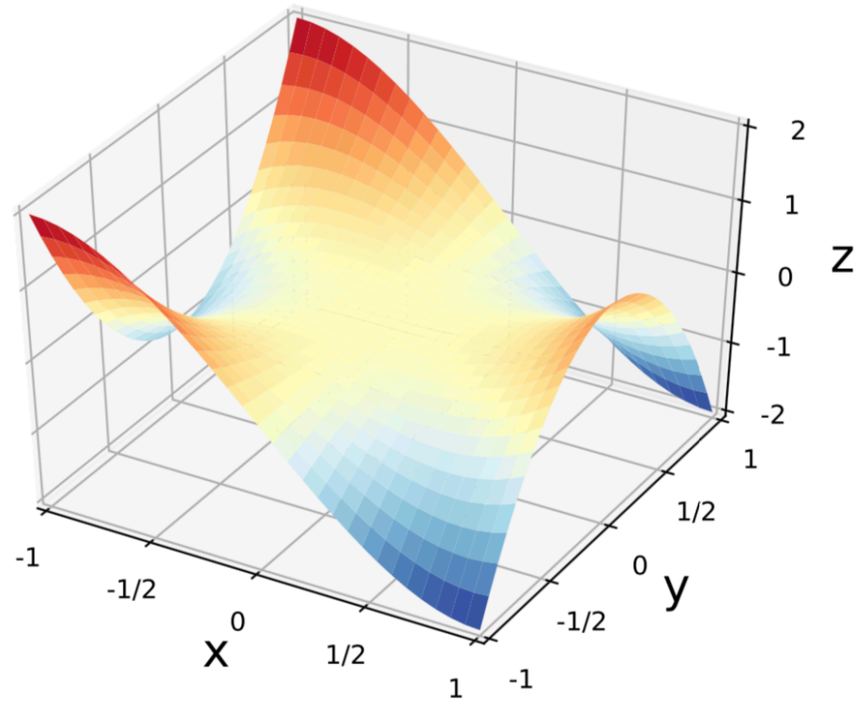
- Study different optimisation algorithms based on GD for simple **1D** (easy) and **2D** functions (more difficult)

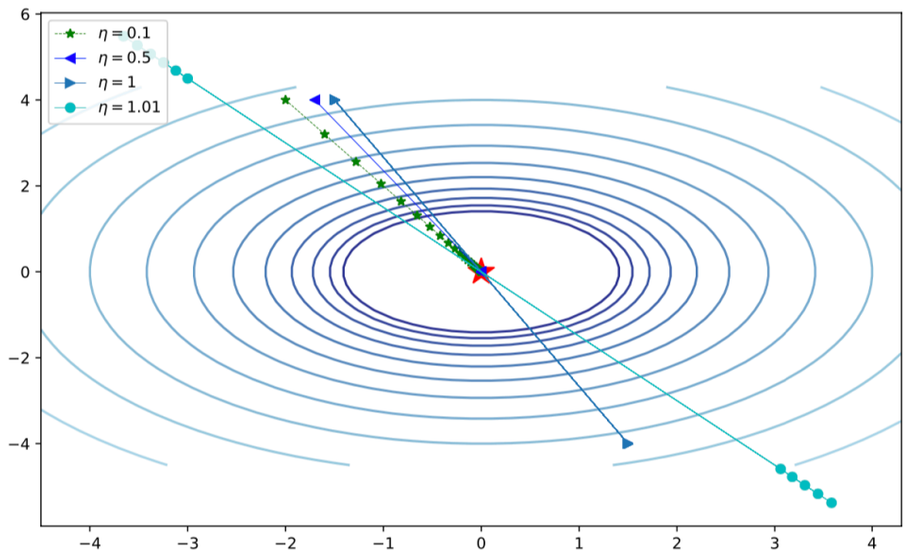
- Visualise the **path of GD algorithms in parameter space**

- Determine which choices of the parameters allow reaching the minima quicker

- Given new 2D functions, show that you can **tune GD to find all the minima**, and compare with the analytical results

challenge: find the most efficient optimiser settings for a given function!





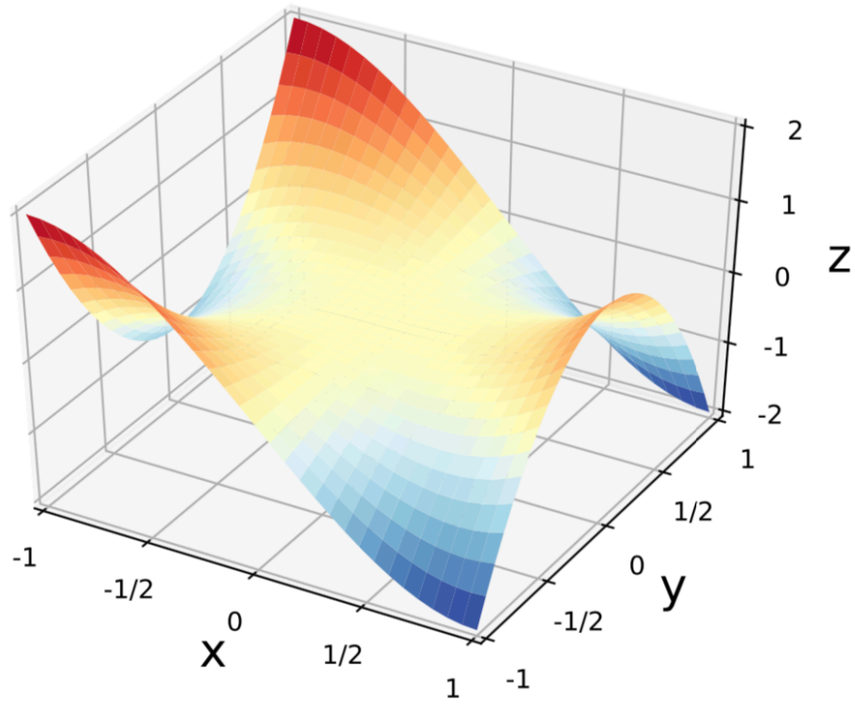
Tutorial 2 (2a & 2b)

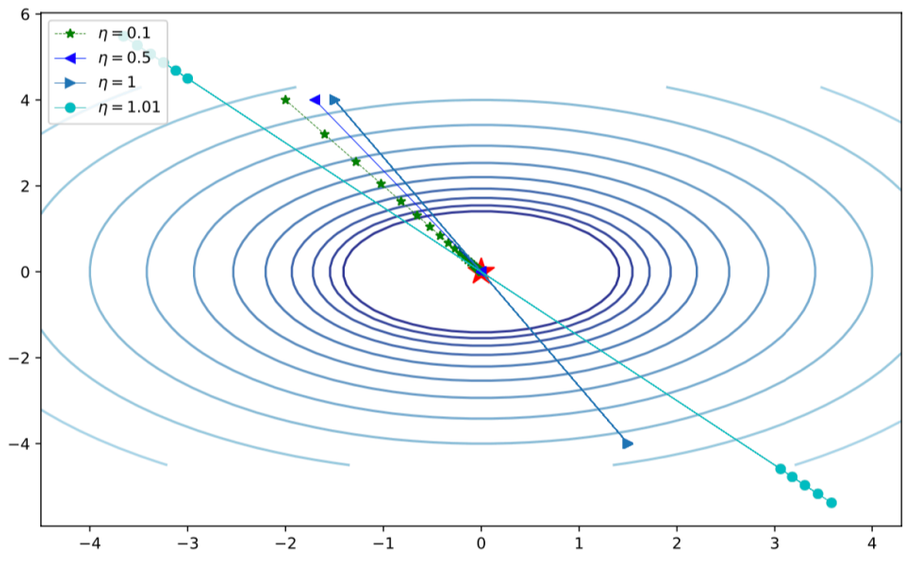
- Study different optimisation algorithms based on GD for simple **1D** (easy) and **2D** functions (more difficult)

- Visualise the **path of GD algorithms in parameter space**

- Determine which choices of the parameters allow reaching the minima quicker

- Given new 2D functions, show that you can **tune GD to find all the minima**, and compare with the analytical results





Question: how could one go beyond hyperparameter setting by trial and error?

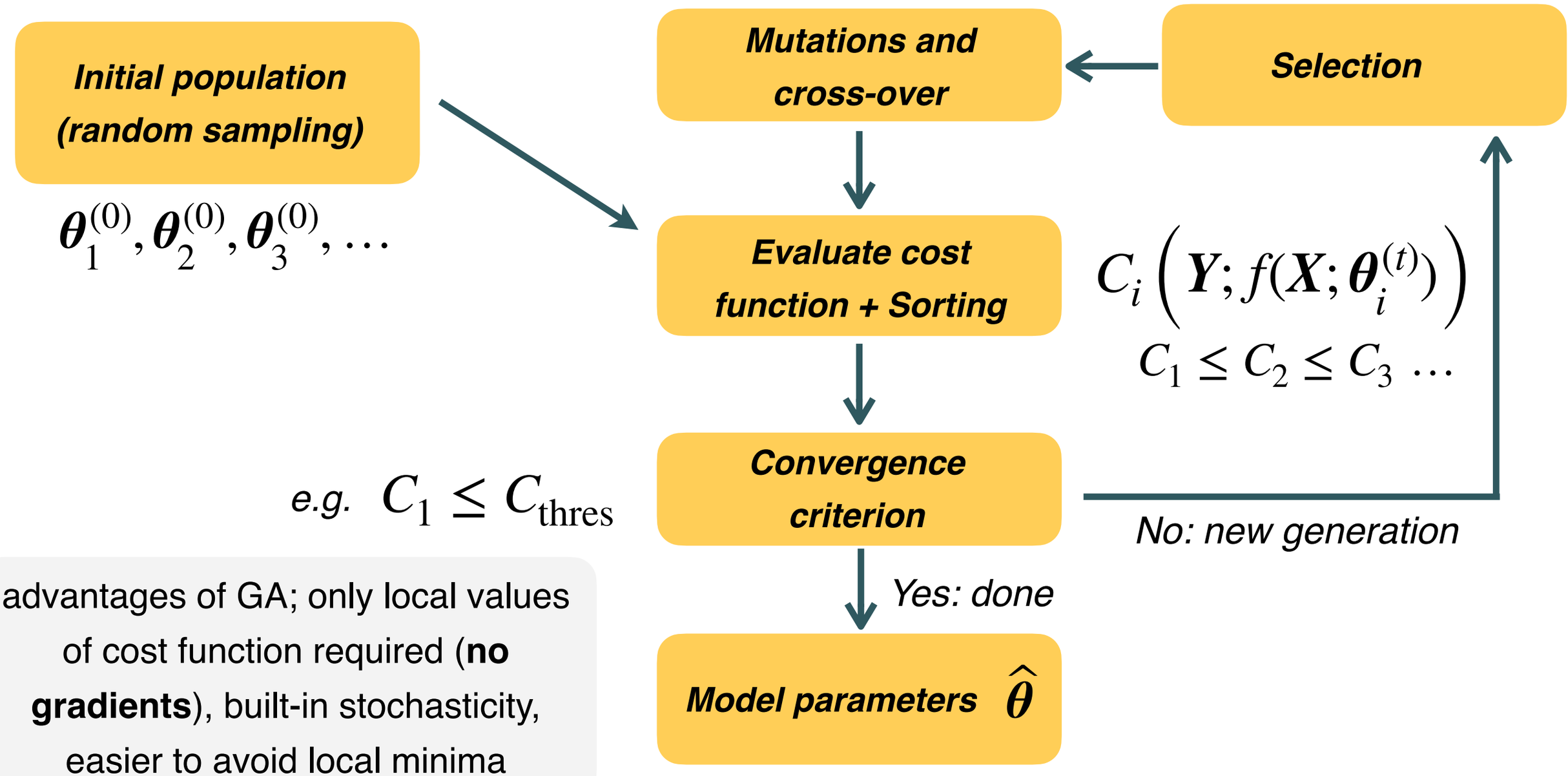
Lecture 3!

Supervised Learning:

Stochastic Optimisation Strategies

Genetic Algorithms

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



e.g. $C_1 \leq C_{\text{thres}}$

advantages of GA; only local values of cost function required (**no gradients**), built-in stochasticity, easier to avoid local minima

Genetic Algorithms

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

mutations

$$\theta_{i,j}^{(t+1)} = \theta_{i,j}^{(t)} \left(1 + \eta_j^{(t)}\right) , \quad j = 1 \ \ldots, m, i = 1, \ldots, M$$

learning rates (depends in generation and parameter)

 # model parameters

 # mutants

crossover

$$\boldsymbol{\theta}_i^{(t)} = \left(\theta_{i,1}^{(t)}, \theta_{i,2}^{(t)}, \ldots, \theta_{i,m}^{(t)}\right) \rightarrow \left(\theta_{i,1}^{(t)}, \theta_{i,2}^{(t)}, \ldots, \theta_{i,p-1}^{(t)}, \theta_{k,p}^{(t)}, \ldots, \theta_{k,m}^{(t)}\right)$$

$$\boldsymbol{\theta}_k^{(t)} = \left(\theta_{k,1}^{(t)}, \theta_{k,2}^{(t)}, \ldots, \theta_{k,m}^{(t)}\right) \rightarrow \left(\theta_{k,1}^{(t)}, \theta_{k,2}^{(t)}, \ldots, \theta_{k,p-1}^{(t)}, \theta_{i,p}^{(t)}, \ldots, \theta_{i,m}^{(t)}\right)$$

children inherit part of the DNA of each parent

Genetic Algorithms

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

mutations

$$\theta_{i,j}^{(t+1)} = \theta_{i,j}^{(t)} \left(1 + \eta_j^{(t)}\right) , \quad j = 1 \ \ldots, m, i = 1, \ldots, M$$

learning rates (depends in generation and parameter)

 # model parameters

 # mutants

crossover

Chromosome1	11011|00100110110
Chromosome2	11011|11000011110
Offspring1	11011|11000011110
Offspring2	11011|00100110110

children inherit part of the DNA of each parent

Genetic Algorithms

we can illustrate how **GAs work in practice** in a graphical way with this example

The CMA-ES algorithm

Improve the efficiency of simple GAs by incorporating global (in addition to local) information on the parameter space

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

- Start by randomly initialising the model parameters using a Gaussian distribution

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

- At every generation, we generate ***M* mutants** according to the following distribution

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

step size

covariance matrix of search distribution

tuned during fit

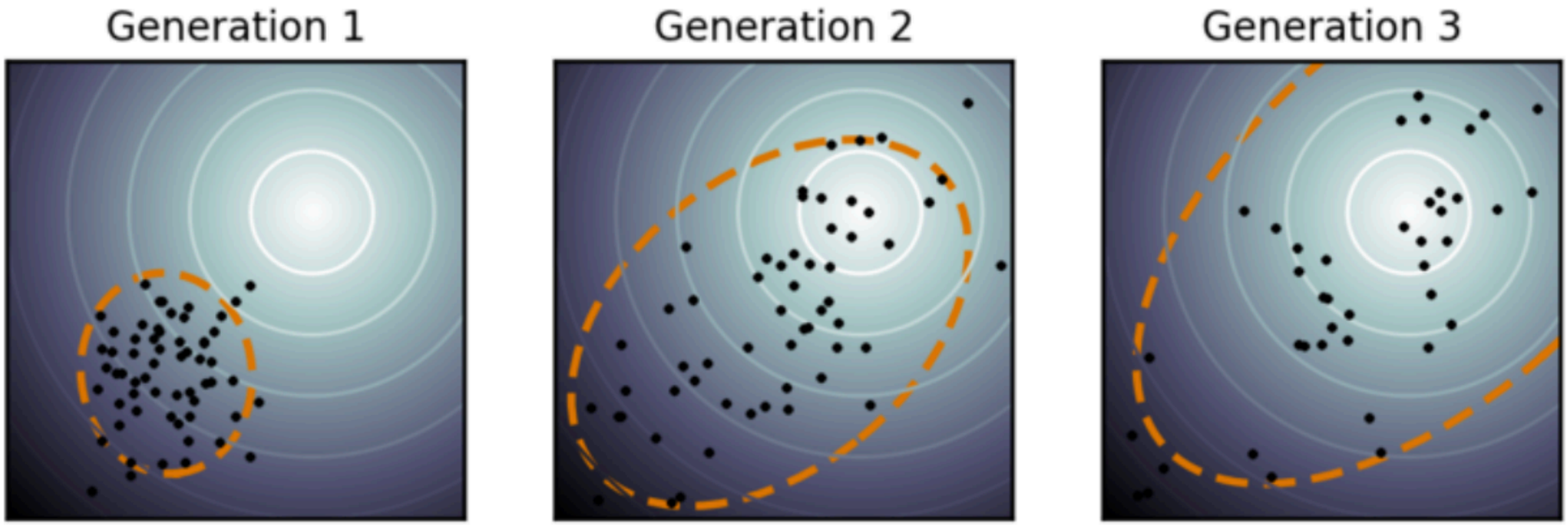
- The **new search centre** is computed as weighted average over fraction of best mutants

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

Parameters of the algorithms

adaptation as the parameter space is explored is key feature of CMA-ES

The CMA-ES algorithm



main advantages: (i) information on gradients never required, only local values of cost function, (ii) information on whole set of mutants (not only best ones) used to tune step size and search covariance matrix

The CMA-ES algorithm applied

The CMA-ES algorithm applied

Optimisation algorithms: summary

- Machine learning problems have associated **high-dimensionality parameter spaces**

- Choice of optimisation algorithm and its hyperparameters crucial for efficient solution of realistic machine learning problems

- Stochastic gradient descent methods are at the core of deep-learning algorithms

- Evolutionary algorithms benefit from built-in stochasticity and can be competitive for problems with very complex space of minima (where multiple quasi-degenerate minima are present)