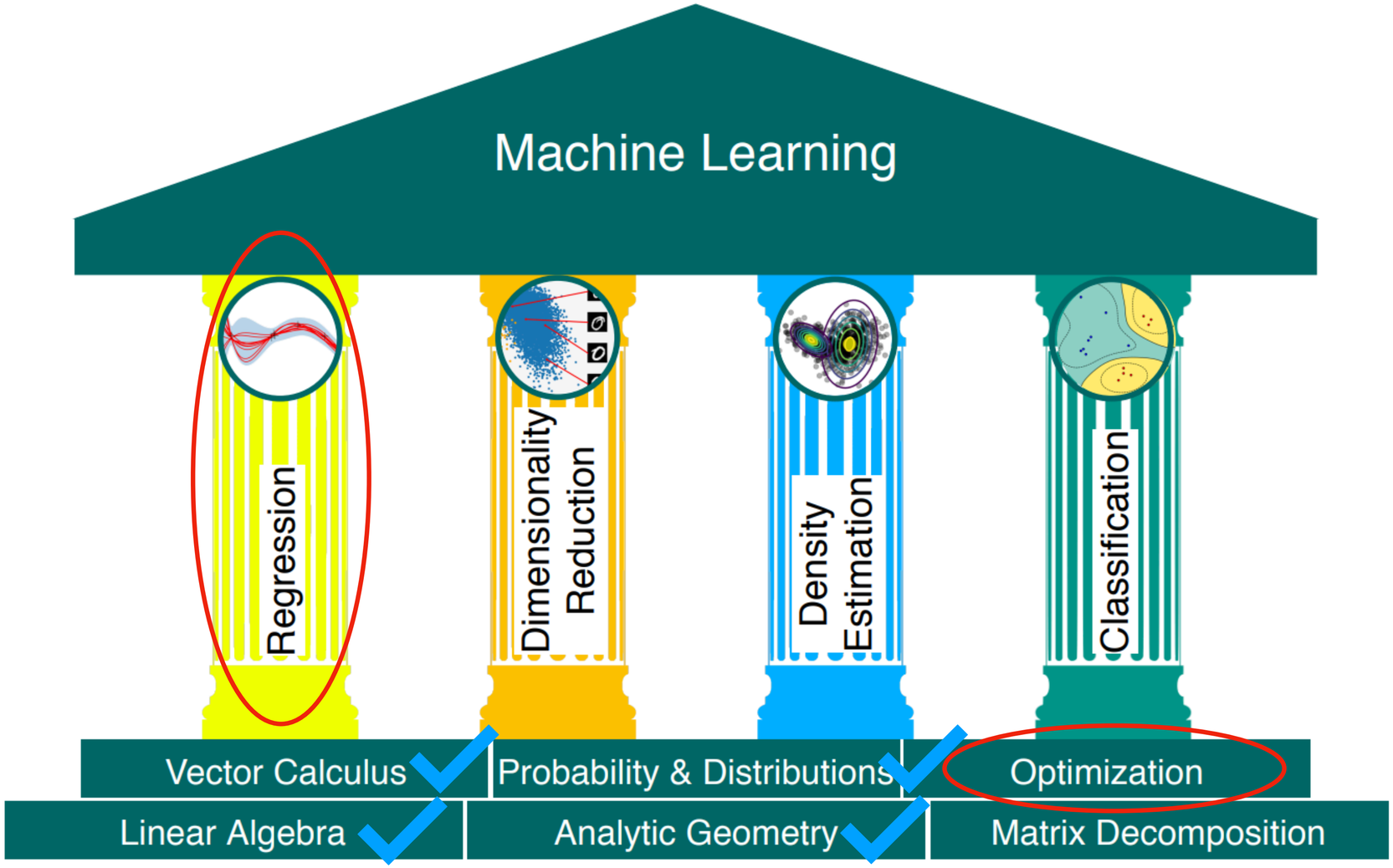
Linear regression

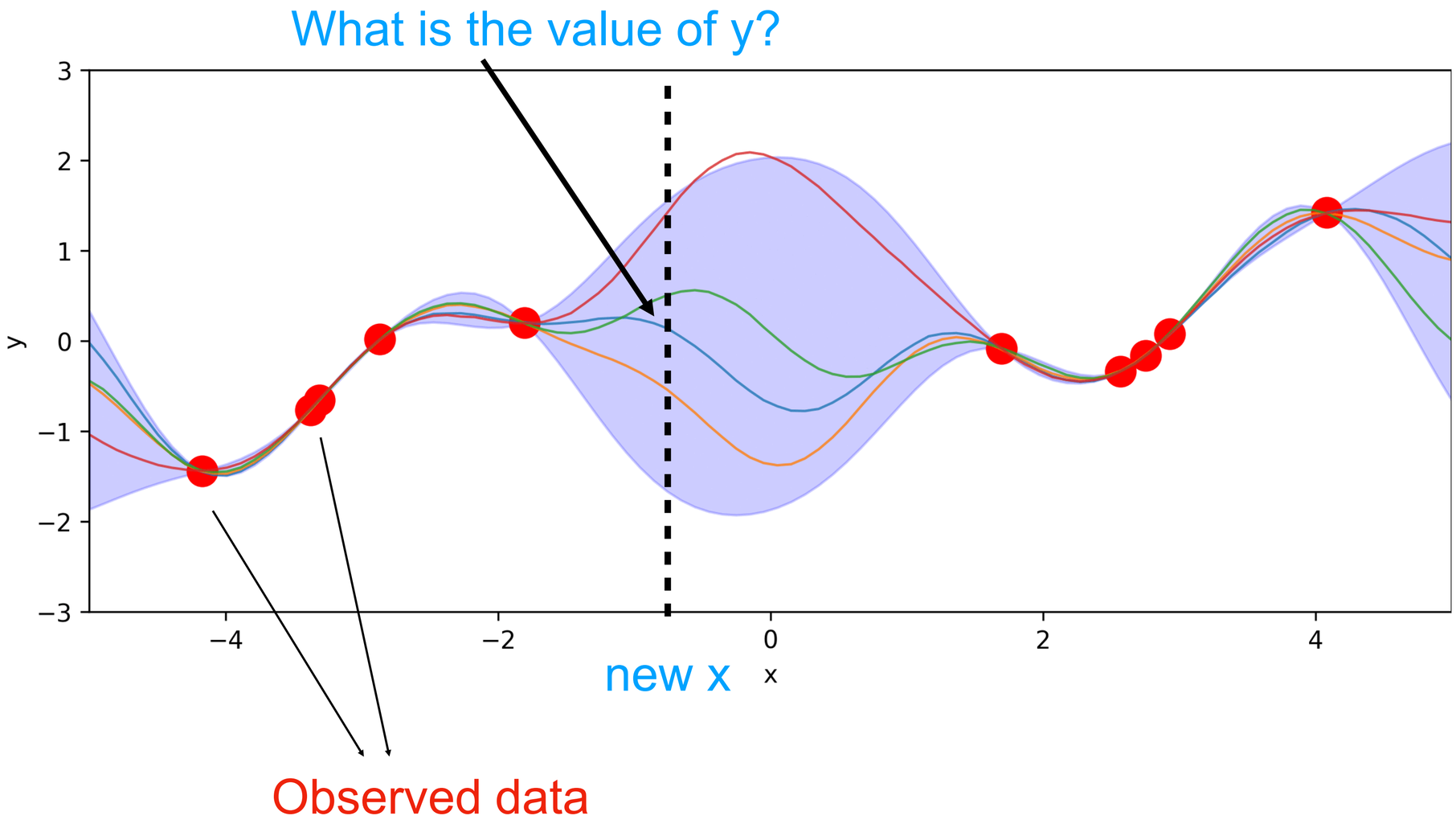
COMP 3670/6670
Rahul Shome
based on slides by Thang Bui

Foundations of ML



Source: Maths for ML Textbook

What is regression?



An example

Scaling Laws for Neural Language Models

Jared Kaplan *
Johns Hopkins University, OpenAI
jaredk@jhu.edu

Sam McCandlish*
OpenAI
sam@openai.com

Tom Henighan
OpenAI
henighan@openai.com

Tom B. Brown
OpenAI
tom@openai.com

Benjamin Chess
OpenAI
bchess@openai.com

Rewon Child
OpenAI
rewon@openai.com

Scott Gray
OpenAI
scott@openai.com

Alec Radford
OpenAI
alec@openai.com

Jeffrey Wu
OpenAI
jeffwu@openai.com

Dario Amodei
OpenAI
damodei@openai.com

How did they get these relationships?

Linear regression

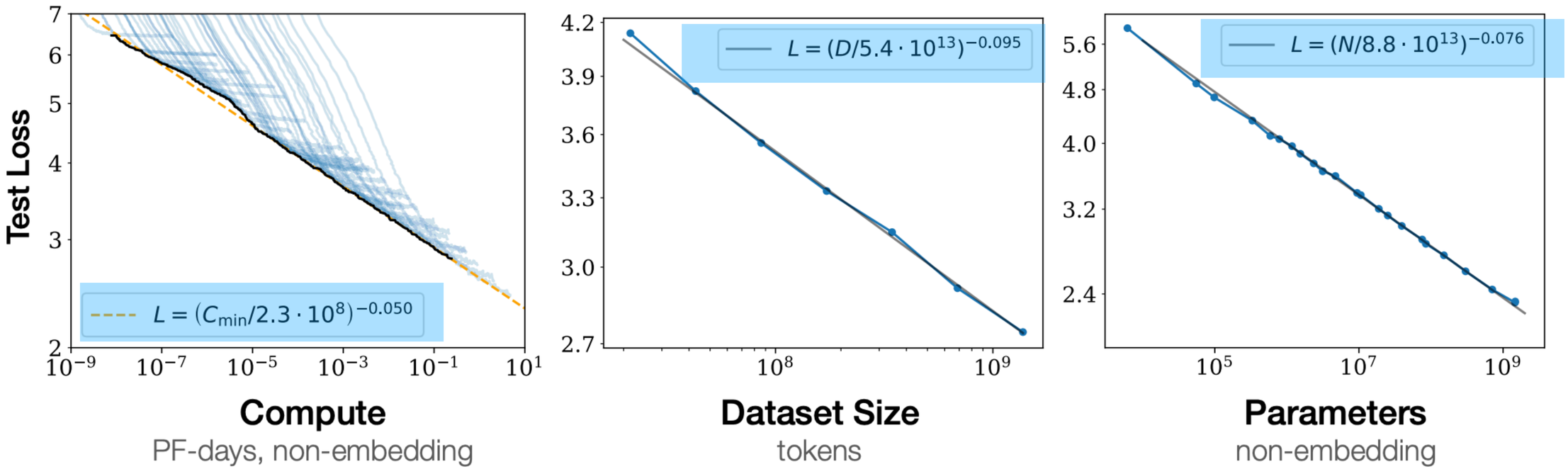


Figure 1 Language modeling performance improves smoothly as we increase the model size, datasetset size, and amount of compute[2] used for training. For optimal performance all three factors must be scaled up in tandem. Empirical performance has a power-law relationship with each individual factor when not bottlenecked by the other two.

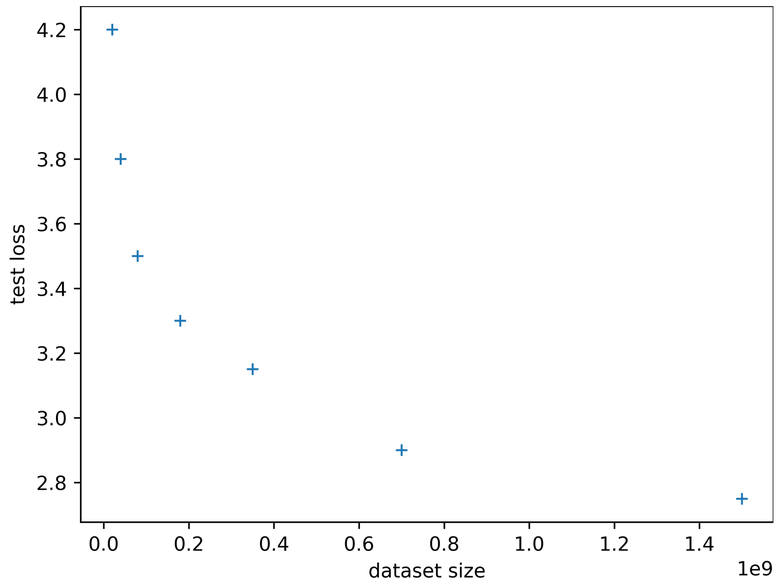
An example - Test loss vs data size [1]

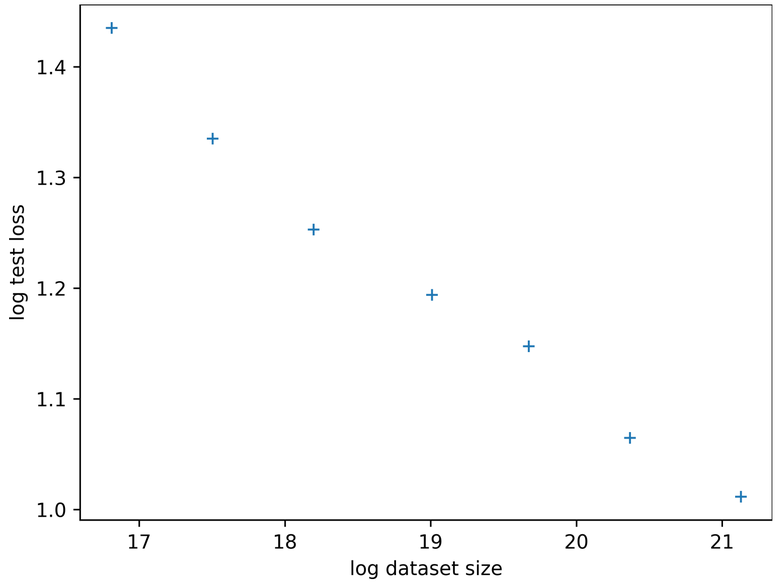
Dataset size	Test loss
2.0e+07	4.2
4.0e+07	3.8
8.0e+07	3.5
1.8e+08	3.3
3.5e+08	3.15
7.0e+08	2.90
1.5e+09	2.75

*

Log transform

log(Dataset size)	log(Test loss)
16.8	1.44
17.5	1.34
18.2	1.25
19.0	1.19
19.7	1.14
20.4	1.06
21.1	1.01





* I got these numbers by eyeballing the plot in the paper

An example - Test loss vs data size [2]

Let x = log(Dataset size), y = log(test loss). Assume $y \approx f(x) = ax + b$

Question: find a and b

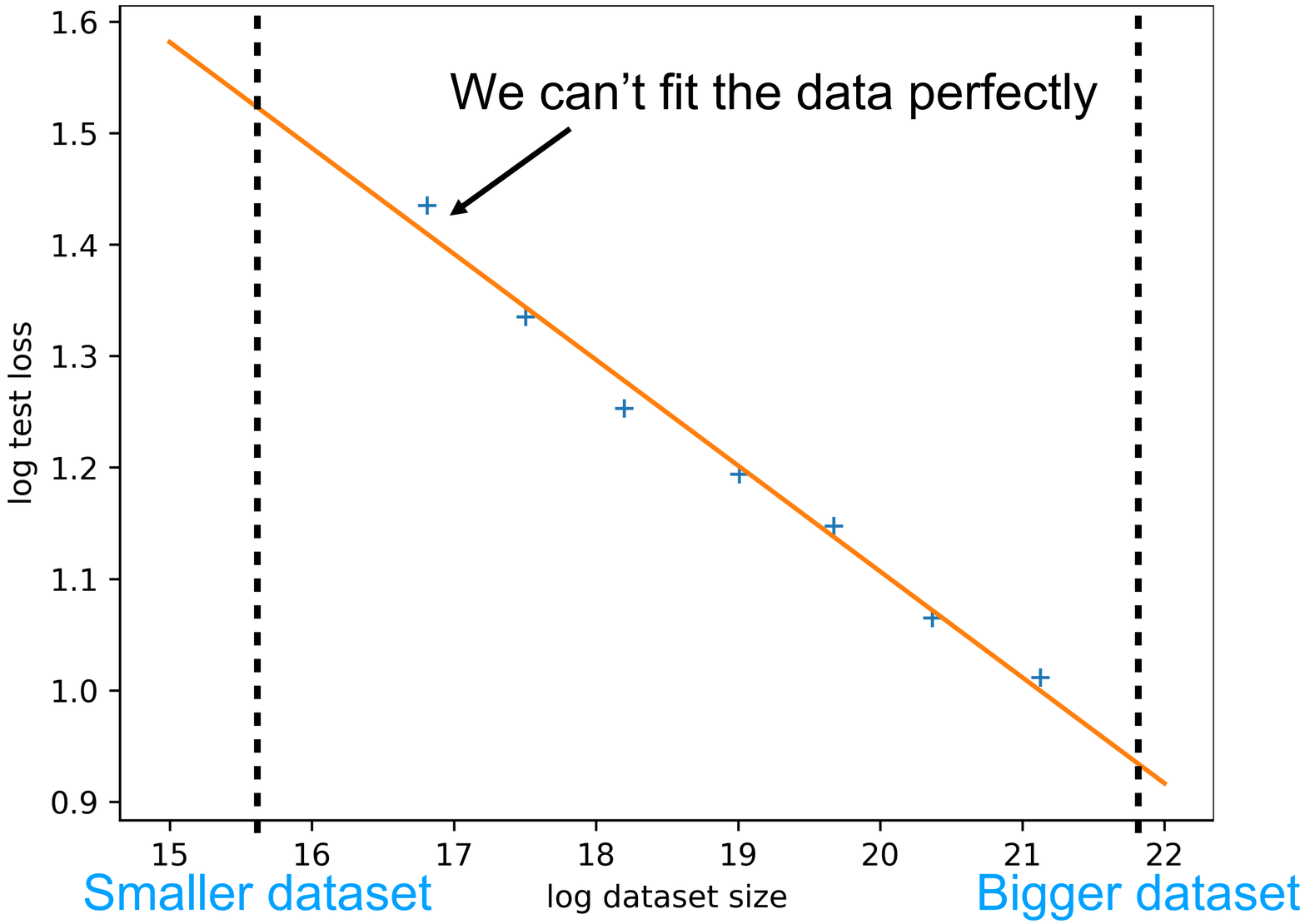
Step 1: Write down an **objective function** *or* **goodness of fit**, which tell us how **good** the current a and b are

Step 2: **Optimise** this objective function

An example - Test loss vs data size [3]

What is the test performance?

What is the test performance?




```
Sx = np.sum(x)
Sy = np.sum(y)
Sxy = np.sum(x * y)
Sx2 = np.sum(x**2)
N = x.shape[0]

a = (N * Sxy - Sx * Sy) / (N * Sx2 - Sx**2)
b = (Sy - a*Sx) / N

xpred = np.linspace(15, 22, 100)
fpred = a * xpred + b
```

$$y = -0.095x + 3$$

$$L = \left(\frac{D}{5 \times 10^{13}}\right)^{-0.095}$$

Data and linear assumption

Training data N input, output pairs $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}, \mathbf{x}_n \in \mathbb{R}^D, y_n \in \mathbb{R}$

Each row is a data point

Each column is a feature dimension

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1D} \\ x_{21} & x_{22} & \cdots & x_{2D} \\ \vdots & \vdots & \vdots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{ND} \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

One scalar per data point

Assumptions:

- Underlying function is **linear**, $f_\theta(\mathbf{x}) = \sum_{d=1}^{D} \theta_d x_d = \theta^\intercal \mathbf{x}, \theta \in \mathbb{R}^D$
- Due to measurement noise, observed y is a noisy version of $f(\mathbf{x})$

$$\begin{bmatrix} f_\theta(\mathbf{x}_1) \\ f_\theta(\mathbf{x}_2) \\ \vdots \\ f_\theta(\mathbf{x}_N) \end{bmatrix} = \begin{bmatrix} \theta^\intercal \mathbf{x}_1 \\ \theta^\intercal \mathbf{x}_2 \\ \vdots \\ \theta^\intercal \mathbf{x}_N \end{bmatrix} = X\theta$$

Test time: given a new input $\mathbf{x}^*$, prediction = $f(\mathbf{x}^*) = \theta^\intercal \mathbf{x}^*$

Question: where is the bias/intercept in this formulation?

Objective function

Training data N input, output pairs $\mathscr{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}, \mathbf{x}_n \in \mathbb{R}^D, y_n \in \mathbb{R}$

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1D} \\ x_{21} & x_{22} & \cdots & x_{2D} \\ \vdots & \vdots & \vdots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{ND} \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \quad \approx \quad \begin{bmatrix} f_\theta(\mathbf{x}_1) \\ f_\theta(\mathbf{x}_2) \\ \vdots \\ f_\theta(\mathbf{x}_N) \end{bmatrix} = \begin{bmatrix} \theta^\intercal \mathbf{x}_1 \\ \theta^\intercal \mathbf{x}_2 \\ \vdots \\ \theta^\intercal \mathbf{x}_N \end{bmatrix} = X\theta$$

Desideratum: y is well approximated by $f_\theta(\mathbf{x}) = \theta^\intercal \mathbf{x}$

Objective function measures the approximation quality. Several options (all smaller is better):

- Raw difference, $y - f_\theta(\mathbf{x})$. What can go wrong?
- Absolute difference, $|y - f_\theta(\mathbf{x})|$. What is the problem here?
- Squared difference, $(y - f_\theta(\mathbf{x}))^2$. Any issue here?

We will use the squared difference, also called L2 loss or squared loss or squared error.

Least squares for linear regression

Loss function: $L(\theta) = \frac{1}{N}\sum_{n=1}^{N}(y_n - f_\theta(\mathbf{x}_n))^2 = \frac{1}{N}\parallel \mathbf{y} - X\theta \parallel_2^2 = \frac{1}{N}(\mathbf{y} - X\theta)^\intercal(\mathbf{y} - X\theta)$

We want to find θ that minimises the loss function. Closed-form analytic solution!

$$\theta = (X^\intercal X)^{-1}X^\intercal\mathbf{y}$$

Try to derive this!

H Matrix

Loss function: $L(\theta) = \frac{1}{N}\sum_{n=1}^{N}(y_n - f_\theta(\mathbf{x}_n))^2 = \frac{1}{N}\parallel \mathbf{y} - X\theta \parallel_2^2 = \frac{1}{N}(\mathbf{y} - X\theta)^\intercal(\mathbf{y} - X\theta)$

Trace of an H Matrix indicates the number of parameters being optimised in model

$$H = X(X^\intercal X)^{-1}X^\intercal$$

Trace: $Tr(X) = \sum_i X_{ii}$

Traces has desirable properties that help us describe matrices.

Check rules of partial derivatives for traces.

*Petersen & Pedersen, The Matrix Cookbook

Linear regression with features

So far, we have discussed linear regression which fits straight lines to data. Fortunately, "linear regression" only refers to "**linear** in the **parameters**".

We can perform an arbitrary nonlinear transformation $\phi(\mathbf{x})$ of the inputs $\mathbf{x}$ and then linearly combine the components. That is, $f_\theta(\mathbf{x}) = \sum_{d=1}^{D} \theta_d \phi(x)_d = \theta^\intercal \phi(\mathbf{x}), \theta \in \mathbb{R}^D$

Loss function: $L(\theta) = \frac{1}{N}\sum_{n=1}^{N}(y_n - f_\theta(\mathbf{x}_n))^2 = \frac{1}{N}\|\mathbf{y} - \Phi\theta\|_2^2 = \frac{1}{N}(\mathbf{y} - \Phi\theta)^\intercal(\mathbf{y} - \Phi\theta)$

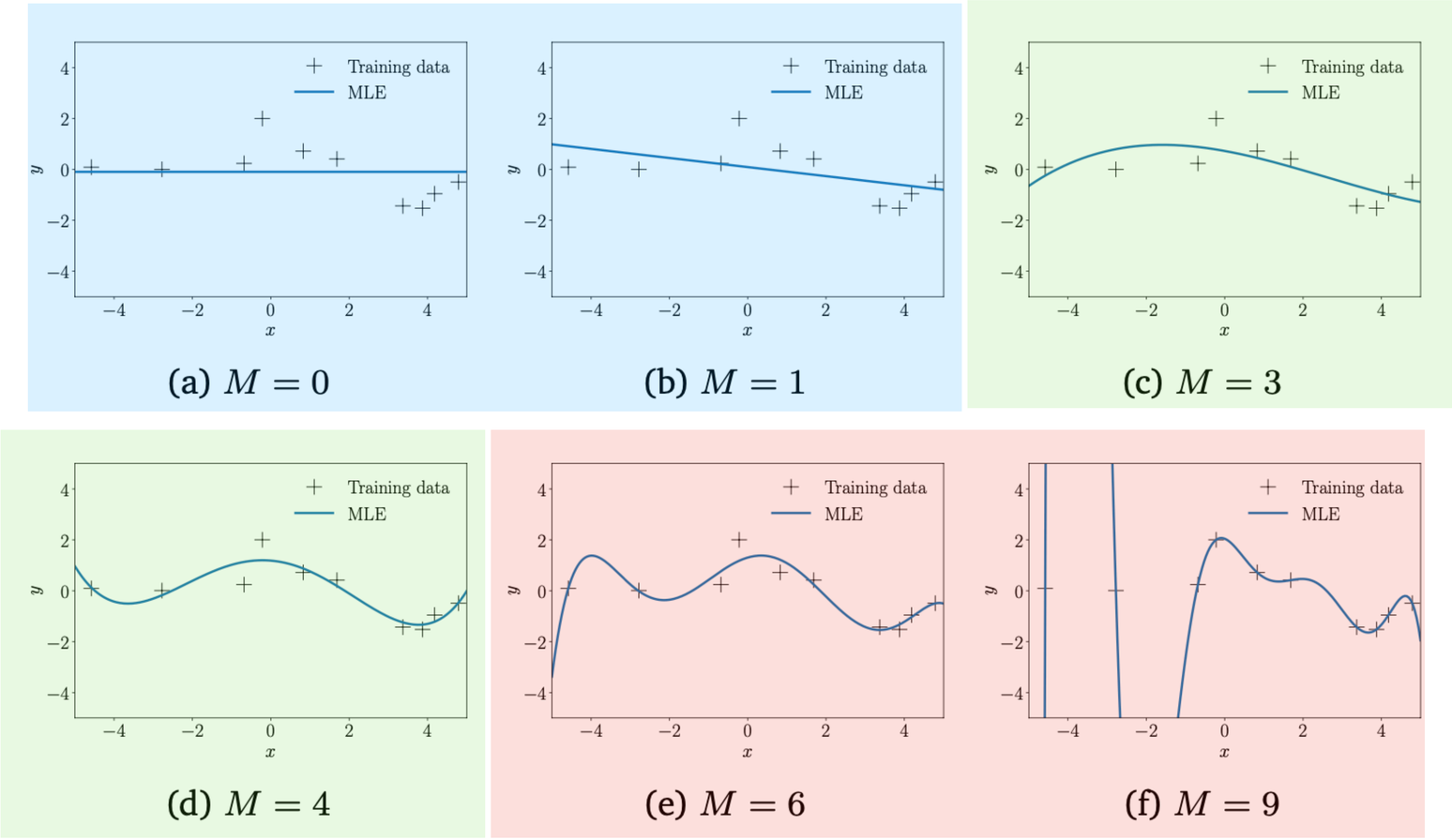
Each row is a data point

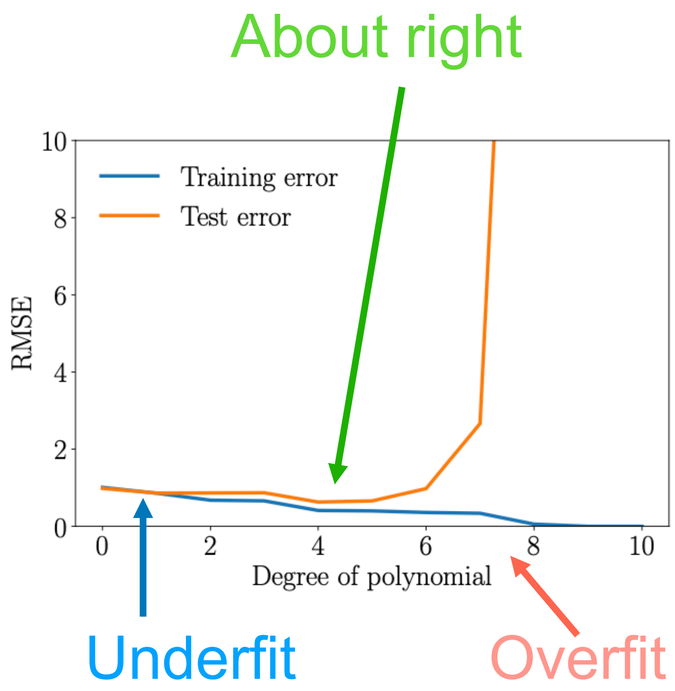
Each col corresponds to a feature mapping

$$\Phi = \begin{bmatrix} \phi_1(x_1) & \phi_2(x_1) & \cdots & \phi_D(x_1) \\ \phi_1(x_2) & \phi_2(x_2) & \cdots & \phi_D(x_2) \\ \vdots & \vdots & \vdots & \vdots \\ \phi_1(x_N) & \phi_2(x_N) & \cdots & \phi_D(x_N) \end{bmatrix}$$

Closed-form analytic solution: $\theta = (\Phi^\intercal\Phi)^{-1}\Phi^\intercal\mathbf{y}$

Polynomial regression example - underfitting and overfitting





Regularised least squares

Overfitting occurs because the model is too **complex** (θ has too many large entries), while there are too limited training sample.

We want to *penalise* the amplitude of parameters by **regularisation.**

Regularised least squares = least squares + regularisation

$$L_\lambda(\theta) = \frac{1}{N}\sum_{n=1}^{N}(y_n - f_\theta(\mathbf{x}_n))^2 + \lambda \parallel \theta \parallel_p^2$$

Data-fit

Lower = better fit

Regulariser

Lower = simpler model

Hyperparameter

We can use any p-norm $\|\cdot\|_p$. Smaller p leads to sparser solutions, i.e., many parameter value

Regularised least squares - analytic solution

Loss function: $L(\theta) = \frac{1}{N} \parallel \mathbf{y} - X\theta \parallel_2^2 + \lambda \parallel \theta \parallel_2^2 = \frac{1}{N}(\mathbf{y} - X\theta)^\intercal(\mathbf{y} - X\theta) + \lambda \parallel \theta \parallel_2^2$

We want to find θ that minimises the loss function. Closed-form analytic solution!

$$\theta = (X^\intercal X + N\lambda \mathrm{I})^{-1} X^\intercal \mathbf{y}$$

Try to derive this!

Linear regression - Potential issues

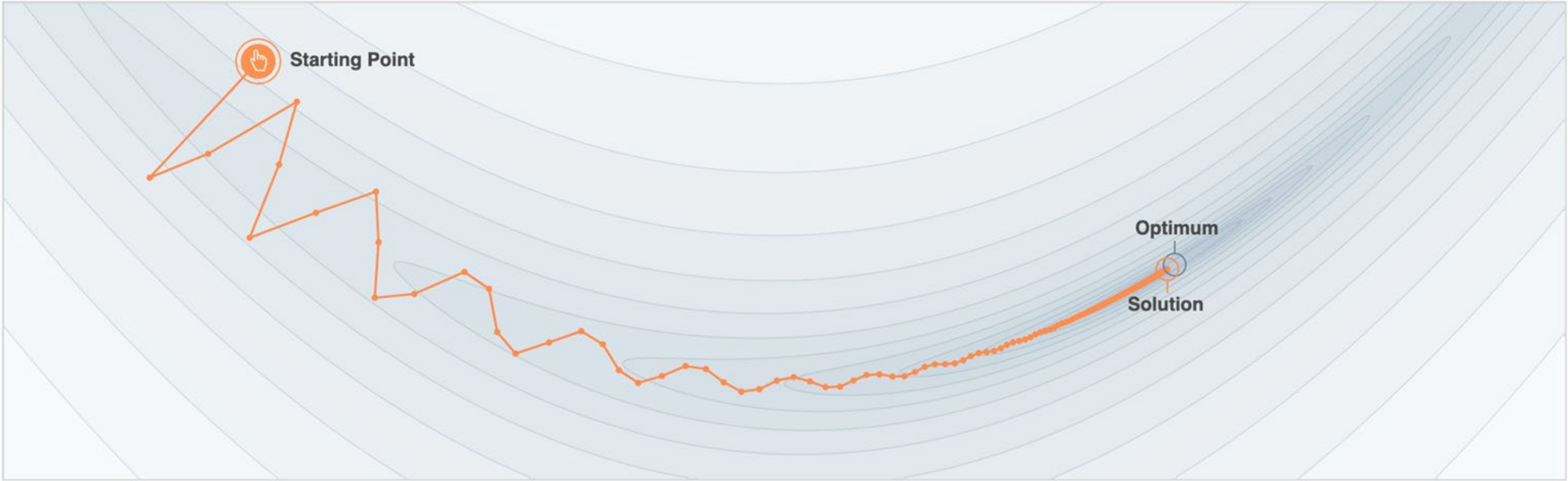
Point estimate $\theta_{\mathrm{ML}} = (X^\intercal X)^{-1} X^\intercal \mathbf{y}$ or $\theta_{\mathrm{MAP}} = (X^\intercal X + N\lambda \mathrm{I})^{-1} X^\intercal \mathbf{y}$

We have assumed this is invertible. But this is not guaranteed! So use this instead! Why?

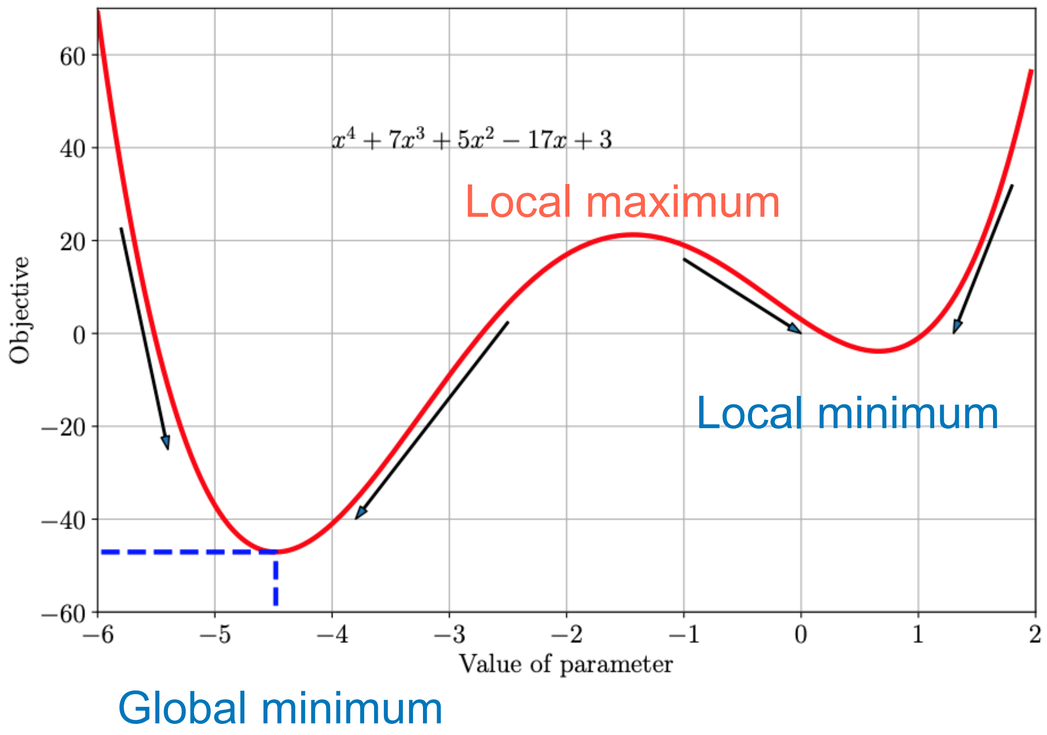
Computational complexity $\mathcal{O}(ND^2 + ND + D^3) = \mathcal{O}(ND^2 + D^3)$. Can be large for large D.

- use matrix inversion lemma, or,
- use numerical optimisation instead

Optimisation



An analytic example



Optimisation using Gradient Descent

- Given $f: \mathbb{R}^D \to \mathbb{R}$, we consider the problem of solving for the minimum of f, $\min f$
- **Gradient descent** is a *first-order* optimisation algorithm.
- To find a local minimum of a function using gradient descent, one takes steps proportional to the **negative of the gradient** of the function at the current point.
- Gradient descent exploits the fact that $f(x_0)$ decreases **fastest** if one moves from x_0 in the direction of the negative gradient $(-\nabla f(x_0))^\intercal$ of f at x_0.
- If $x_1 = x_0 - \gamma(\nabla f(x_0))^\intercal$, for a small step size $\gamma \geq 0$, then $f(x_1) \leq f(x_0)$
- Example: https://distill.pub/2017/momentum/

Gradient descent - step size heuristic

Choosing a good step-size (learning rate) is important in gradient descent

- If the step-size is too small, gradient descent can be slow
- If the step-size is chosen too large, gradient descent can overshoot, fail to converge, or even diverge

There are several heuristics to adapt the step size

- When the function value increases after a gradient step, the step-size was too large. Undo the step and decrease the step-size
- When the function value decreases the step could have been larger. Try to increase the step-size.
- Heuristically, we choose a learning rate that starts big and ends small, e.g., $\gamma_i = 1/(i+1)$

Gradient descent with momentum

- The convergence of gradient descent may be very slow if the curvature of the optimization surface is such that there are regions that are poorly scaled
- The proposed method to improve convergence is to give gradient descent some memory
- Gradient descent with momentum is a method that introduces an additional term to remember what happened in the previous iteration.
- This memory dampens oscillations and smooths out the gradient updates
- The idea is to have a gradient update with memory to implement a moving average

$$x_{i+1} = x_i - \gamma_i m_i$$
$$m_i = (1-\alpha)m_{i-1} + \alpha(\nabla f(x_i))^{\intercal}, \alpha \in [0,1]$$