Contents

1 Numerical Computation

1.1 Real numbers representation

Theorem (Base representation). *A non-zero real number in base $\beta \geq 2$ can be uniquely expressed as:*

$$x = sign(x) \cdot (d_1\beta^{-1} + d_2\beta^{-2} + \ldots) \cdot \beta^p = sign(x) \cdot m \cdot \beta^p$$

Here, $sign(x) = 1$ if $x > 0$, $sign(x) = -1$ if $x < 0$, p is an integer and the digits $d_1, d_2, d_3, \ldots$ satisfy the following conditions:

- $0 \leq d_i \leq \beta - 1$
- $d_1 \neq 0$

There are two ways to represent a number:

- **Normalized scientific representation**:

$$x = \pm(0.d_1d_2d_3\ldots)\beta^p$$

- **Mixed representation**:

$$x = \pm d_1d_2d_3\ldots d_p.d_{p+1}d_{p+2}\ldots \quad p > 0$$

$$x = \pm 0.0\ldots 0d_1d_2\ldots \quad p > 0$$

Example

$x = (12.751)_{10} = +(0.12751) \cdot 10^2$

Definition (Finite numbers). *The set of finite numbers, denoted as $\mathcal{F}(\beta, t, L, U)$, consists of the numbers expressed in base β with t digits, and the exponent is constrained within the range L to U. In this context, any floating point number $x \in \mathcal{F}(\beta, t, L, U)$ has the form:*

$$fl(x) = \pm(d_1 + \ldots + d_t)\beta^p \quad L \leq p \leq U$$

Here, $m = (d_1 \ldots d_t)$ is called mantissa and each d_i is an integer satisfying the condition:

$$0 \leq d_i < \beta \quad i = 1, \ldots, t$$

Example

$x = (12.\bar{3})_{10}, \quad \mathcal{F}(10, 5, -3, 3), \quad \text{fl}(x)?$

$$0.12333 \cdot 10^2$$

This set is finite and discrete and its cardinality is given by:

$$\#\mathcal{F}(\beta, t, L, U) = 2(\beta - 1)\beta^{t-1}(U - L + 1)$$

Within this set, the minimum value is given by:

$$\min(\mathcal{F}(\beta, t, L, U)) = \beta^{L-1}$$

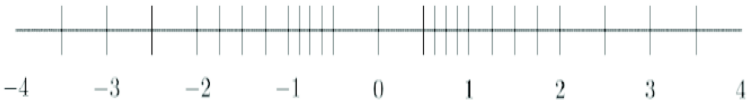
and the maximum value is given by:

$$\max(\mathcal{F}(\beta, t, L, U)) = \beta^U(1 - \beta^{-t})$$

it's worth noting that floating-point numbers are equally spaced only between successive powers of β. They have a higher density around 0, with density diminishing as they move further from 0.

Example

Example $\mathcal{F}(2, 3, -1, 1)$



- Number of elements: $2(2-1)2^2(1+1+1)+1 = 25$
- UFL $= 2^{-2} = 0.25$
- OFL $= 2^1(1-2^{-3}) = 1.75$

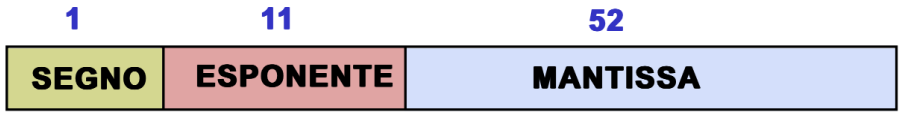
Not all real numbers are representable exactly, so it's necessary to assign an approximate finite number using these two rules:

- **Chop**: truncate the β base representation of x after the t-st digit.

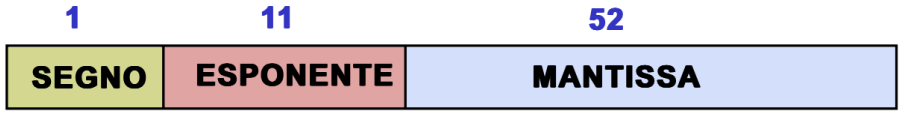
- **Round to nearest**: $\text{fl}(x)$ is the nearest floating-point to x. In case the real number is equally distant from two consecutive finite numbers, we round to the even number (this is called round to even)

Finite numbers are represented on computers using a standard denoted as **Standard IEEE**, which enables code portability between different processors. There are two main specifications:

- **Single Precision**: 32 bits, with 1 for the sign, 23 for the mantissa, and 8 for the exponent.



- **Double Precision**: 64 bits, with 1 for the sign, 52 for the mantissa, and 11 for the exponent.



Since the computer's base is two, and for any non-zero number the first digit is always equal to 1, it is consider implicit and not stored. This allows for 24 bits in the case of single precision and 53 bits for double precision.

Definition (Machine Precision)**.** *Let $\mathcal{F}(\beta, t, L, U)$ be a set of finite numbers, and the unit roundoff (or machine precision), denoted by ϵ, is defined as follows:*

$$\epsilon = \begin{cases} \beta^{1-t} & \textit{With rounding by chopping} \\ \frac{1}{2}\beta^{1-t} & \textit{With rounding to nearest} \end{cases}$$

In the standard IEEE, where rounding to the nearest in the chosen rounding rule, the machine precision ϵ is given by:

- $\epsilon_{\text{single}} \approx 10^{-8}$
- $\epsilon_{\text{double}} \approx 10^{-16}$

Theorem. *The maximum relative error in representing real number x within range of floating-point system is given by*

$$\left| \frac{fl(x) - x}{x} \right| \leq \epsilon$$

The IEEE standard introduces special values to indicate two exceptional situations:

- `Inf`, which stands for *infinity* results from dividing a finite number by zero, such as $1/0$.
- `NaN`, which stands for *not a number* results from undefined or indeterminate operations, such as $0/0$.

These special values are implemented through reserved representations.

1.2 Finite arithmetic

Numerical result from computer program inherently contain errors, which can be categorized as follows:

- **Measure errors**: due to the measure instrument.
- **Algorithmic errors**: due to the individual operations within the algorithmic process.
- **Truncation errors**: when an infinite procedure is approximated with a finite one.
- **Inherent errors**: due to the finite representation of data.

Let $\tilde{x}$ be an approximation obtained through an algorithm for the true value x. The error can be measured using:

- **Absolute error**: $E_x = \tilde{x} - x$
- **Relative error**: $R_x = \frac{\tilde{x}-x}{x}, \quad x \neq 0$

Definition (Floating-point Operation)**.** *For two numbers $x, y \in \mathcal{F}$ the operation is defined as*

$$x \ \tilde{op} \ y = fl(x \ op \ y)$$

This means that the firstly the operation is done in exact arithmetic, and subsequently, the result is rounded to fit in the given finite set of numbers.
The exact operation is done on a register situated near the processor, equipped with additional bits.

Each operation causes an error, which is given by:

$$\left| \frac{(x \ \tilde{op} \ y) - (x \ op \ y)}{x \ op \ y} \right|$$

The addition and subtraction of two t-digit numbers can lead to a phenomenon known as **cancellation**. This one occurs when either (a) the two numbers significantly differ magnitude or (b) they have opposite sign while having similar magnitudes. Such occurrences may result in a loss of information, giving a risk of error propagation that could potentially invalidate the final result.

Example

$\mathcal{F}(10, 6, L, U)$
$x = 0.147554326,\ \text{fl}(x) = 0.147554$
$y = -0.147251742,\ \text{fl}(y) = 0.147525$

$$x + y = 0.147554326 - 0.1475251742 = 0.302584 \times 10^{-3}$$

$$x \tilde{+} y = 0.147554 - 0.1475251 = 0.302000 \times 10^{-3}$$

$$\frac{(x \tilde{+} y) - (x + y)}{(x + y)} = \frac{0.302000 \times 10^{3} - 0.302584 \times 10^{-3}}{0.302584 \times 10^{-3}} = 0.2 \times 10^{-2}$$

Which is way higher than the $\epsilon = \frac{1}{2}10^{1-6} = 0.5 \times 10^{-5}$.

2 Linear Algebra Tools

This chapter introduces inner product to give geometric meaning to vectors and vector spaces, enabling calculations of length, distance, and angles.

Definition (Symmetric Positive Definitive Matrix)**.** *A symmetric matrix $A \in \mathbb{R}^{n\times n}$ that satisfies*

$$\text{for every nonzero vector } x : x^T Ax > 0 \tag{2.1}$$

*is called **positive definite**. If only $\geq$ holds in 2.1, then A is called **positive semidefinite**.*

These properties helps in identifying positive definite matrices without having to check the definition explicitly:

1. The null space of A contains only the null vector;
2. The diagonal elements a_{ii} of A are positive;
3. The eigenvalues of A are real and positive.

2.1 Angles and Orthogonality

The angle ω between vectors x and y is computed as:

$$\cos\omega = \frac{\langle x, y\rangle}{\|x\|_2\|y\|_2}$$

Here, $\langle x, y\rangle$ denotes the inner product between x and y.
This angle indicated the vectors' similarity in orientation.

Definition (Orthogonal vectors)**.** *Two vectors are orthogonal if $\langle x, y\rangle = 0$. If additionally $\|x\| = 1 = \|y\|$, then x and y are orthonormal.*

Definition (Orthogonal matrix)**.** *A square matrix is an orthogonal matrix if and only if its columns are orthonormal so that*

$$AA^T = I = A^T A$$

which implies that

$$A^{-1} = A^T$$

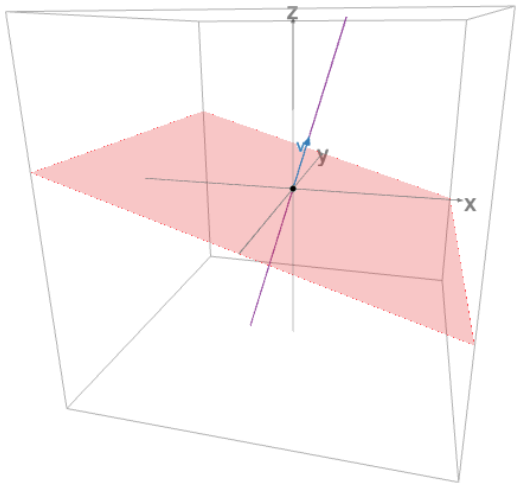
The length of a vector x is not changed when transforming it using an orthogonal matrix A.

$$\|Ax\|_2^2 = \|x\|_2^2$$

Moreover, the angle between any two vectors x, y is also unchanged when transforming both of them using an orthogonal matrix A.

Definition (Orthonormal Basis)**.** *In an n-dimensional vector space V with a basis set $\{b_1, \ldots, b_n\}$, if all the basis vectors are orthogonal to each other, the basis is called as an **orthogonal basis**. Additionally, if the length of each basis vector is 1, the basis is referred to as an **orthonormal basis**.*

We can also have vector spaces that are orthogonal to each other. Given a vector space V of dimension D, let's consider a subspace U of dimension M such that $U \subseteq V$. Then its **orthogonal complement** $U^\perp$ is a $D - M$ dimensional subspace V and contains all vectors in V that are orthogonal to every vector in U.



2.1.1 Orthogonal Projections

Projections are key linear transformations in machine learning and are particularly useful for handling high-dimensional data. Often, only a few dimensions in such data are essential for capturing the most relevant information. By projecting the original high-dimensional data onto a lower dimensional feature space, we can work more efficiently to learn about the dataset and extract significant patterns.

Definition (Projection)**.** *Let V be a vector space and $U \subseteq V$ a subspace of V. A linear mapping $\pi : V \to U$ is called **projection** if it satisfies $\pi^2 = \pi \circ \pi = \pi$.*

Given that linear mappings can be represented by transformation matrices, the above definition extends naturally to *projection matrices* P_π. These matrices exhibit the property that $P_\pi^2 = P_\pi$.
The projection $\pi_U(x)$ of a vector $x \in \mathbb{R}^n$ onto a subspace U is the closest point necessarily in U to x.

3 Matrix Decompositions

3.1 Eigenvalues and Eigenvectors

Eigenanalysis helps us understand linear transformations represented by a matrix A. Eigenvectors x are special vectors that only get scaled, not rotated, when multiplied by A. The scaling factor is the eigenvalue λ, which indicated how much x is stretched or shrunk. λ can also be zero.

Definition (Eigenvalue and Eigenvector)**.** *Let $A \in \mathbb{R}^{n\times n}$ be a square matrix. Then $\lambda \in \mathbb{R}$ is an **eigenvalue** of A and nonzero vector x is the corresponding **eigenvector** of A if*

$$Ax = \lambda x \tag{3.1}$$

*We call 3.1 the **eigenvalue equation**.*

The following statements are equivalent:

- λ is an eigenvalue of $A \in \mathbb{R}^{n\times n}$.
- A nonzero vector x exists such that $Ax = \lambda x$ or, equivalently, $(A - \lambda I_n)x = 0$ for $x \neq 0$.
- Then $A - \lambda I$ is a **singular matrix** and its determinant is **zero**.

Each eigenvector x has one unique eigenvalue λ, but each λ can have multiple eigenvectors.

Definition (Eigenspace and Eigenspectrum)**.** *For $A \in \mathbb{R}^{n\times n}$, the set of all eigenvectors of A associated with an eigenvalue λ spans a subspace of $\mathbb{R}^n$, which is called the **eigenspace** of A with respect to λ and is denoted by E_λ. The set of all eigenvalues of A is called the **eigenspectrum** of A.*

Definition. *Let λ_i be an eigenvalue of a square matrix A. Then the **geometric multiplicity** of λ_i is the number of linearly independent eigenvectors associated with λ_i. In other words, it is the dimensionality of the eigenspace spanned by the eigenvectors associated with λ_i.*

Theorem. *The eigenvectors $x_1, \ldots, x_n$ of a matrix $A \in \mathbb{R}^{n\times n}$ with n distinct eigenvalues $\lambda_1, \ldots, \lambda_n$ are linearly independent.*

This theorem states that eigenvectors of a matrix with n distinct eigenvalues form a basis of $\mathbb{R}^n$.

4 Least Square Problem

Considering the problem

$$\underset{m\times n}{A}\ x = b$$

with more rows than columns ($m > n$).
Finding an exact solution to this system, i.e. satisfying $Ax - b = 0$ is impossible. However, we can seek an approximate solution $\tilde{x}$, such that $A\tilde{x} - b \approx 0$.
To quantify the closeness of the vector $A\tilde{x} - b$, also called residual , to zero, we use the 2-norm. Thus,

$$\tilde{x} = \arg\min_{x\in\mathbb{R}^n} \|Ax - b\|_2$$

This minimization problem can be expressed in terms of the squared 2-norm, which is equivalent:

$$\tilde{x} = \arg\min_{x\in\mathbb{R}^n} \|Ax - b\|_2^2$$

Considering the $r = \text{rank}(A)$ we have two possibilities:

1. $r = n \rightarrow$ The least square problem has a unique solution $\forall b \in \mathbb{R}^m$.

   By solving the so-called **normal equations**

   $$A^T Ax = A^T b$$

   where $A^T A$ is $n \times n$ symmetric and positive definite matrix if $\text{rank}(A) = n$ then the matrix is non singular. This can be computed efficiently using the direct method Cholesky decomposition.

   For any B symmetric and positive definite, the Cholesky decomposition factorizes it as the product of two matrices:

   $$B = LL^T$$

   where L is lower triangular and L^T is upper triangular.

   The computational complexity is $\mathcal{O}(\frac{n^3}{6})$, which is half of complexity of LU factorization.

   The solution x is then computed as

   $$A^T Ax = A^T b$$
   $$L\underbrace{L^T x}_{y} = A^T b$$
   $$\begin{cases} Ly = A^T b \\ L^T x = y \end{cases}$$

2. $r < n \rightarrow$ The least square problem has infinite solutions $\forall b \in \mathbb{R}^m$.

   Among the infinite possible solutions $S = \{x \in \mathbb{R}^n \mid x \text{ solutions of } \min\|Ax - b\|_2^2\}$ we are interested in computing one unique x^* that has the minimum norm (2-norm):

   $$x^* = \min_{x\in S} \|x\|_2$$

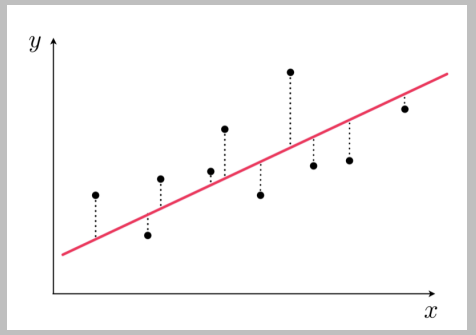
   That is computed using the SVD decomposition of $A = U\Sigma V^T$:

   $$x^* = \sum_{i=1}^{r} \frac{u_i^T b}{\sigma_i} v_i$$

   where $U = (u_1, \ldots, u_m) \in \mathbb{R}^{m\times m}$, $V = (v_1, \ldots, v_n) \in \mathbb{R}^{n\times n}$, $\sigma_i \in \mathbb{R}^+$ and $b \in \mathbb{R}^m$.

Example

Consider a polynomial of degree n given by $f_\theta(x) = \theta_0 + \ldots + \theta_n x^n$, where we want compute the parameters $\theta = \{\theta_0, \ldots, \theta_n\}$ that fit well the given data pairs $(x_i, y_i)_{i=0,\ldots,m}$.



Fitting the data means to minimize the residuals (the distance between an observed value and the predictor's value) $r = (r_0, \ldots, r_m) = y - Ax - b$, through the solution of a least square problem. In this context, $Ax - b$ is defined as follows:

$$\begin{cases} r_0 = y_0 - f(x_0) = y_0 - (\theta_0 + \ldots + \theta_n x_0^n) \\ \vdots \\ r_m = y_m - f(x_m) = y_m - (\theta_0 + \ldots + \theta_n x_m^n) \end{cases}$$

This expression can be more compactly rewritten as $r = y - A\theta$ where A is given

by:

$$A = \begin{bmatrix} 1 & x_0 & x_0^2 & \dots & x_0^n \\ 1 & x_1 & x_1^2 & \dots & x_1^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_m & x_m^2 & \dots & x_m^n \end{bmatrix}$$

Therefore, the task of minimizing r can be formulated as:

$$\min_{\theta \in \mathbb{R}^n} \|y - A\theta\|_2^2$$

5 Vector calculus

Firstly, we'll explore partial derivatives and gradients, focusing on functions that take a vector as input and produce a single real number as output. These functions are formally represented as $f : \mathbb{R}^n \to \mathbb{R}$.
Subsequently, we will extend these ideas to functions that not only take a vector as input but also produce a vector as output. These functions can be written as $f : \mathbb{R}^n \to \mathbb{R}^m$.

5.1 Gradients of Real-Valued Functions

When we deal with a function that depends on multiple variables, such as $f(x) = f(x_1, x_2)$, we use the **gradient** to represent its derivative. The gradient is a vector composed of **partial derivates** of the function. To compute each partial derivates, we differentiate the function with respect to one variable while keeping all other variables constant.

$$\nabla_x f = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} & \cdots & \frac{\partial f}{\partial x_n} \end{bmatrix} \in \mathbb{R}^{1\times n} \tag{5.1}$$

where n is the number of variables.

Basic Rules of Partial Differentiation

Product rule:
$$\frac{\partial}{\partial x}(f(x)g(x)) = \frac{\partial f}{\partial x}g(x) + f(x)\frac{\partial g}{\partial x}$$

Sum rule:
$$\frac{\partial}{\partial x}(f(x) + g(x)) = \frac{\partial f}{\partial x} + \frac{\partial g}{\partial x}$$

Chain rule:
$$\frac{\partial}{\partial x}(g \circ f)(x) = \frac{\partial}{\partial x}\left(g(f(x))\right) = \frac{\partial g}{\partial f}\frac{\partial f}{\partial x}$$

In the context of the chain rule, consider f as implicitly a composition $f \circ g$.
If a function $f(x_1, x_2)$ is a function of x_1 and x_2, where $x_1(t)$ and $x_2(t)$ are themselves functions of a single variable t, the chain rule yields the partial derivates

$$\frac{\mathrm{d}f}{\mathrm{d}t} = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} \end{bmatrix} \begin{bmatrix} \frac{\partial x_1(t)}{\partial t} \\ \frac{\partial x_2(t)}{\partial t} \end{bmatrix} = \frac{\partial f}{\partial x_1}\frac{\partial x_1}{\partial t} + \frac{\partial f}{\partial x_2}\frac{\partial x_2}{\partial t}$$

Example

Consider $f(x_1, x_2) = x_1^2 + 2x_2$, where $x_1 = \sin t$ and $x_2 = \cos t$, then

$$\text{with} \quad \frac{\partial f}{\partial x_1} = 2x_1, \quad \frac{\partial f}{\partial x_2} = 2$$

$$\begin{aligned} \frac{\mathrm{d}f}{\mathrm{d}t} &= 2\sin t\frac{\partial \sin t}{\partial t} + 2\frac{\partial \cos t}{\partial t} \\ &= 2\sin t\cos t - 2\sin t \end{aligned}$$

If a function $f(x_1, x_2)$ is a function of x_1 and x_2, where $x_1(s, t)$ and $x_2(s, t)$ are themselves functions of two variables s and t, the chain rule yields the partial derivates

$$\frac{\mathrm{d}f}{\mathrm{d}(s,t)} = \begin{bmatrix} \frac{\partial f}{\partial s} & \frac{\partial f}{\partial t} \end{bmatrix}$$

where

$$\begin{aligned} \frac{\partial f}{\partial s} &= \frac{\partial f}{\partial x_1}\frac{\partial x_1}{\partial s} + \frac{\partial f}{\partial x_2}\frac{\partial x_2}{\partial s} \\ \frac{\partial f}{\partial t} &= \frac{\partial f}{\partial x_1}\frac{\partial x_1}{\partial t} + \frac{\partial f}{\partial x_2}\frac{\partial x_2}{\partial t} \end{aligned}$$

Another way to obtain these two partial derivatives is to represent the previous formula as a row vector containing the partial derivatives of f with respect to x_1 and x_2. This row vector is then multiplied by a matrix composed of the partial derivatives of x_1 and x_2 with respect to s and t. When you perform this multiplication, you get the exact same result as above.

$$\begin{bmatrix} \frac{\partial f}{\partial s} & \frac{\partial f}{\partial t} \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} \end{bmatrix} \begin{bmatrix} \frac{\partial x_1}{\partial s} & \frac{\partial x_1}{\partial t} \\ \frac{\partial x_2}{\partial s} & \frac{\partial x_2}{\partial t} \end{bmatrix}$$

Example

Given the following functions:
$g : \mathbb{R}^2 \to \mathbb{R}^2 \quad g(s,t) = (\sin(t)s, \cos(s)t)$
$f : \mathbb{R}^2 \to \mathbb{R} \quad f(x_1, x_2) = x_1^2 + 2x_2$
$f \circ g : \mathbb{R}^2 \to \mathbb{R}$
Compute $\nabla_{(s,t)}(f \circ g)$ and evaluate $\nabla_{(s,t)}(f \circ g)(0,0)$.

$$= \begin{bmatrix} 2s\sin(t) & 2 \end{bmatrix} \begin{bmatrix} \sin(t) & s\cos(t) \\ -t\sin(s) & \cos(s) \end{bmatrix}$$
$$= \begin{bmatrix} 2s\sin^2(t) - 2t\sin(s) \\ 2s^2\sin(t)\cos(t) + 2\cos(s) \end{bmatrix} = (0, 2)$$

5.2 Gradients of Vector-Valued Functions

We can express a vector-valued function $f : \mathbb{R}^n \to \mathbb{R}^m$ as a column vector of m real-valued functions $f_i : \mathbb{R}^n \to \mathbb{R}$. Given an input vector $x = [x_1, \ldots, x_n]^T \in \mathbb{R}^n$, the output is defined as:

$$f(x) = \begin{bmatrix} f_1(x) \\ \vdots \\ f_m(x) \end{bmatrix} \in \mathbb{R}^m$$

Definition (Jacobian). *By contrast, in Equation 5.1, each partial derivative $\frac{\partial f}{\partial x_i}$ is a column vector.*

$$J = \nabla_x f = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \cdots & \frac{\partial f}{\partial x_n} \end{bmatrix}$$
$$= \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$
$$J(i,j) = \frac{\partial f_i}{\partial x_j}$$

*The collection of all first-order partial derivatives of a vector-valued function $f : \mathbb{R}^n \to \mathbb{R}^m$ is called the **Jacobian**. The Jacobian J is an $m \times n$ matrix.*

Definition (Hessian). *The **Hessian**, denoted as $\nabla^2_{x_i,x_j}$, is the collection of all second-order derivatives and the corresponding **Hessian matrix***

$$H = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix} \in \mathbb{R}^{n \times n}$$

is symmetric. The element at position i, j is defined as $\frac{\partial^2 f}{\partial x_i x \partial x_j}$.

Example

$$f : \mathbb{R}^2 \to \mathbb{R}^3, \quad f : (x_1, x_2) = \begin{pmatrix} x_1 + x_2 \\ 2x_1^2 - x_2 \\ -x_1 x_2 \end{pmatrix}$$
$$J(f) : \mathbb{R}^2 \to \mathbb{R}^{3\times 2}, \quad J(i,j) = \begin{bmatrix} 1 & 1 \\ 4x_1 & -1 \\ -x_2 & -x_1 \end{bmatrix}, \quad J(1,1) = \begin{bmatrix} 1 & 1 \\ 4 & -1 \\ -1 & -1 \end{bmatrix}$$

Example

Let us consider the linear model

$$y = \Phi\theta$$

where $\theta \in \mathbb{R}^D$ is a parameter vector, $\Phi \in \mathbb{R}^{N\times D}$ are input features, and $y \in \mathbb{R}^N$ are the corresponding observations. We define the functions

$$e : \mathbb{R}^D \to \mathbb{R}^N, \quad e(\theta) = y - \Phi\theta$$
$$L : \mathbb{R}^N \to \mathbb{R}, \quad L(e) = \|e\|_2^2, \quad L(\theta) = \|y - \Phi\theta\|_2^2$$

This is called a **least-squares loss** function.
We want to find $\frac{\partial L}{\partial \theta}$, which is derivative of the loss function with respect to the parameters θ. This will allow us to find the optimal θ that minimizes the loss function $L(\theta)$.
The chain rule allows us to compute the gradient as

$$\frac{\partial L}{\partial e} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial \theta}$$

We know that $\|e\|_2^2 = e^T e$ and so

$$\frac{\partial L}{\partial e} = 2e^T \in \mathbb{R}^{1\times N}$$

Furthermore, we obtain

$$\frac{\partial e}{\partial \theta} = -\Phi \in \mathbb{R}^{N\times D}$$

such that our desired derivative is

$$\nabla L_\theta = -2e^T\Phi = -2\underbrace{(y^T - \theta^T\Phi^T)}_{1\times N}\underbrace{\Phi}_{N\times D} \in \mathbb{R}^{1\times D}$$

5.3 Backpropagation and Automatic Differentiation

In machine learning, finding optimal model parameters often involves performing gradient descent. This requires computing the gradient of a learning objective with respect to the model's parameters. Calculating the gradient explicitly can be impractical due to the complexity and length of the resulting derivative equations. To address this, the **backpropagation** algorithm was introduced in 1962 as an efficient way to compute these gradients, particularly for neural networks.
In neural networks, the output y is computed through a multi-layered function composition $y = (f_K \circ f_{K-1} \circ \cdots f_1)(x)$. Here, x are the inputs (e.g., images), y are the observations (e.g., class labels). Each functions $f_i, i = 1, \ldots, K$, has its own parameters. Specifically, in the i^{th} layer, the function is given $f_i(x_{i-1}) = \sigma(A_{i-1} + b_{i-1})$, where x_{i-1} is the output from layer $i-1$ and σ is an activation function.

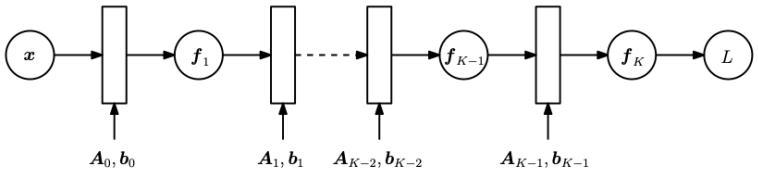


Figure 5.2 Forward pass in a multi-layer neural network to compute the loss L as a function of the inputs $\boldsymbol{x}$ and the parameters $\boldsymbol{A}_i,\ \boldsymbol{b}_i$.

In order to train a neural network, we aim to minimize a loss function L with respect to all parameters A_j, b_j for $j = 0, \ldots, K-1$. Specifically, we're interested in optimizing these parameters to minimize the squared loss given by

$$L(\theta) = \|y - f_K(\theta, x)\|^2$$

where $\theta = \{A_0, b_0, \ldots, A_{K-1}, b_{K-1}\}$.
To minimize $L(\theta)$ we need to compute its gradients of L to the parameter set θ. This involes calculating the partial derivatives of L with respect to the parameters $\theta_j = \{A_j, b_j\}$ for each layer $j = 0, \ldots, K-1$. The chain rule allows us to determine the partial derivatives as

$$\frac{\partial L}{\partial \theta_{K-1}} = \frac{\partial L}{\partial f_K}\frac{\partial f_K}{\partial \theta_{K-1}}$$
$$\frac{\partial L}{\partial \theta_{K-2}} = \frac{\partial L}{\partial f_K}\boxed{\frac{\partial f_K}{\partial f_{K-1}}\frac{\partial f_{K-1}}{\partial \theta_{K-2}}}$$
$$\frac{\partial L}{\partial \theta_{K-3}} = \frac{\partial L}{\partial f_K}\frac{\partial f_K}{\partial f_{K-1}}\boxed{\frac{\partial f_{K-1}}{\partial f_{K-2}}\frac{\partial f_{K-2}}{\partial \theta_{K-3}}}$$
$$\frac{\partial L}{\partial \theta_i} = \frac{\partial L}{\partial f_K}\frac{\partial f_K}{\partial f_{K-1}}\cdots\boxed{\frac{\partial f_{i+2}}{\partial f_{i+1}}\frac{\partial f_{i+1}}{\partial \theta_i}}$$

The red terms are partial derivatives of the output of a layer with respect to its inputs, whereas the blue terms are partial derivatives of the output of a layer with respect to its parameters. The additional terms that we need to compute are indicated by the boxes.
The key insight of backpropagation is to reuse previously computed derivatives to avoid redundant calculations. When we've computed the partial derivatives $\frac{\partial L}{\partial \theta_{i+1}}$, we can reuse them to efficiently calculate the partial derivatives $\frac{\partial L}{\partial \theta_i}$.
It turns out that backpropagation is a special case of a set of techniques known as **automatic differentiation**. Automatic differentiation numerically evaluate the exact (up to machine precision) gradient of a function by working with intermediate variables and applying the chain rule.

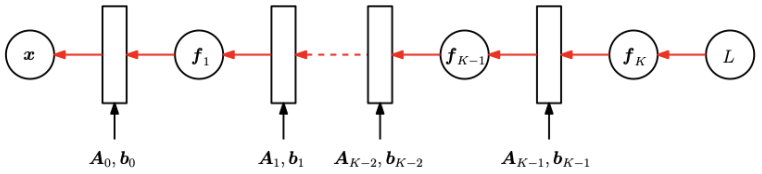


Figure 5.2 Backward pass in a multi-layer neural network to compute the gradients of the loss function.

Example

Consider the real-valued function

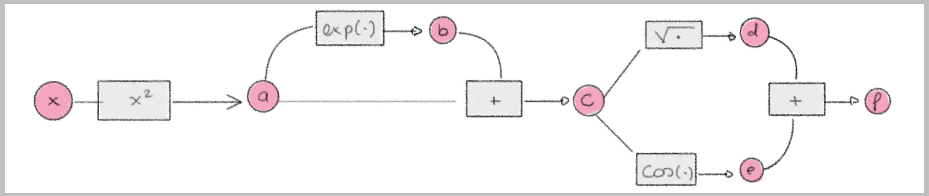
$$f(x) = \sqrt{x^2 + \exp(x^2)} + \cos(x^2 + \exp(x^2))$$

Another way to attach this would be to just define some *intermediate variables*.

Say

$$\begin{aligned} a &= x^2 \\ b &= \exp(a) \\ c &= a + b \\ d &= \sqrt{c} \\ e &= \cos(c) \\ f &= d + e \end{aligned}$$

The set of equations that include intermediate variables can be thought of as a computational graph



By looking at the computation graph, we can compute $\frac{\partial f}{\partial x}$ by working backward from the end of the graph and obtain the derivative of each variable, making the use of the derivatives of the children of that variable

$$\begin{aligned} \frac{\partial f}{\partial d} &= \frac{\partial f}{\partial e} = 1 \\ \frac{\partial f}{\partial c} &= \frac{\partial f}{\partial d}\frac{\partial d}{\partial c} + \frac{\partial f}{\partial e}\frac{\partial e}{\partial c} \\ \frac{\partial f}{\partial b} &= \frac{\partial f}{\partial c}\frac{\partial c}{\partial b} \\ \frac{\partial f}{\partial a} &= \frac{\partial f}{\partial b}\frac{\partial b}{\partial a} + \frac{\partial f}{\partial c}\frac{\partial c}{\partial a} \\ \frac{\partial f}{\partial x} &= \frac{\partial f}{\partial a}\frac{\partial a}{\partial x} \end{aligned}$$

We observe that the computation required for calculating the derivative is of similar complexity as the computation of the function itself (forward pass).

Automatic differentiation is a formalization of last Example. Let $x_1, \ldots, x_d$ be the input variables to the function, $x_{d+1}, \ldots, x_{D-1}$ be the intermediate variables, and x_D the output variable. Then the computation graph can be expressed as follows:

$$\text{For } i = d+1, \ldots, D: \quad x_i = g_i(x_{Pa}(x_i)) \tag{5.2}$$

where the $g_i(\cdot)$ are elementary functions and $x_{Pa}(x_i)$ are the parent nodes of the variable x_i in the graph.

Recall that by definition $f = x_D$ and hence

$$\frac{\partial f}{\partial x_D} = 1$$

For other variables x_i, we apply the chain rule

$$\frac{\partial f}{\partial x_i} = \sum_{x_j : x_i \in Pa(x_j)} \frac{\partial f}{\partial x_j}\frac{\partial x_j}{\partial x_i} = \sum_{x_j : x_i \in Pa(x_j)} \frac{\partial f}{\partial x_j}\frac{\partial g_j}{\partial x_i} \tag{5.3}$$

where $Pa(x_j)$ is the set of parent nodes of x_j in the computation graph. In other words, we apply the chain rule to any node for which x_i is a parent, and so on. Equation 5.2 is the forward pass, whereas 5.3 is the backward pass.
The automatic differentiation approach works whenever we have a function that can be expressed as a computation graph, where the elementary functions are differentiable.

6 Continuous Optimization

Training a machine learning essentially involves identifying a good set of parameters. What constitutes "good" is defined by the objective function. Optimization algorithms are employed to locate the best possible value of this function. Typically, the aim is to minimize the objective function, implying that the best value is the minimum one.

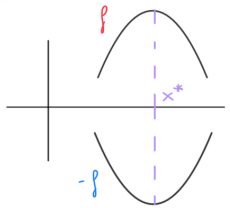


Figure 1: $\arg\max_{x\in\mathbb{R}^n} f(x) = \arg\min_{x\in\mathbb{R}^n} -f(x)$

We will assume in this chapter that our objective function $f : \mathbb{R}^n \to \mathbb{R}$ is differentiable, hence we have access to a gradient to help us find the optimal value. Intuitively, finding the best value is like finding the valleys of the objective function, and the gradients point us uphill. The idea is to move downhill (opposite to the gradient) and hope to find the deepest point.

6.1 Conditions for the existence of the minimum

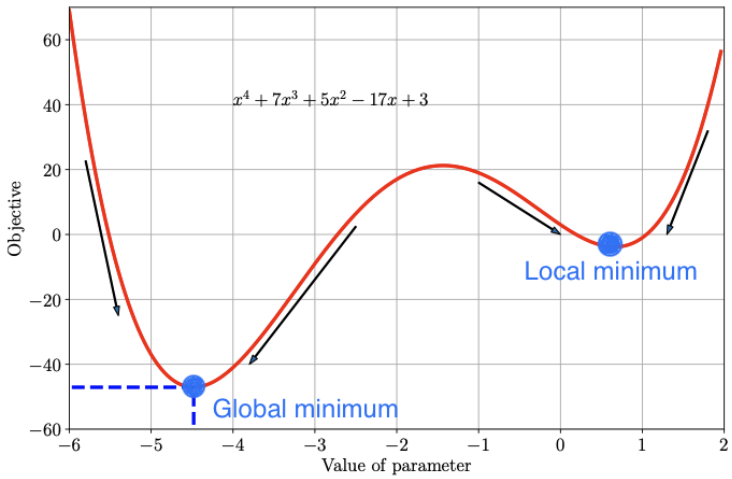
Definition. *f is differentiable if the partial derivates $\frac{\partial f}{\partial x_i}, i = 1, \dots, n$ exist and are continuous.*

Definition. *$x^* \in \mathbb{R}^n$ is a (strict) **local minimum** of f if there exists $\epsilon > 0$ such that:*

$$f(x^*)(<) \leq f(x) \quad \forall x : \|x - x^*\| < \epsilon$$

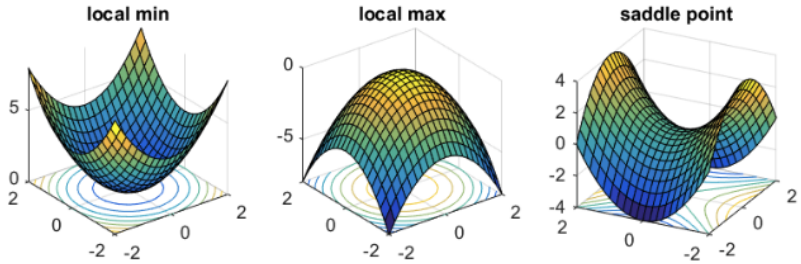
Definition. *$x^* \in \mathbb{R}^n$ is a (strict) **global minimum** of f if*

$$f(x^*)(<) \leq f(x) \quad \forall x \in \mathbb{R}^N$$



Definition (First order conditions)**.** *If x^* is a minimum point of f, then $\nabla f(x^*) = 0$. Furthermore, if $\nabla f(x^*) = 0$ for $x^* \in \mathbb{R}^n$, then x^* can be either a (local) minimum, a (local) maximum or a saddle point of $f(x)$.*

Consequently, we want to find a point $x^* \in \mathbb{R}^n$ such that $\nabla f(x^*) = 0$. Those points are stationary points for f.



Definition (Second order conditions)**.** *if f is twice differentiable and if $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ (the hessian of f) is positive definitive then x^* is a strict local minimum for f.*

6.2 Algorithm to compute the minimum

Iterative methods. Given an initial vector $x_0 \in \mathbb{R}^n$, it is possible to approximate a solution to a given optimization problem by applying iterative methods. In particular, by using these methods, one can compute x_{k+1} as follows:

$$x_{k+1} = g(x_k)$$

until convergence. In this case g is an arbitrary function. Using these methods, $x_k \to x^*$ for $k \to \infty$, where x^* is a stationary point.

Descent methods are iterative methods in which one can compute x_{k+1} as follows:

$$x_{k+1} = x_k + \alpha_k p_k$$

where $p_k \in \mathbb{R}^n$ and $\alpha_k \in \mathbb{R}$.

Definition. *p_k is called a **descent direction** for f in x if there exists $\alpha_k > 0$ such that:*

$$f(x_k + \alpha_k p_k) < f(x_k)$$

In this case:

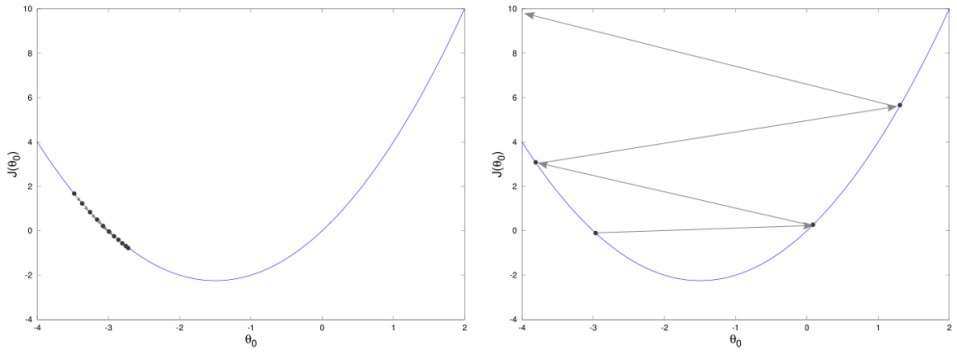
$$p_k^T \nabla f(x_k) < 0 \quad \text{if } p \neq 0$$

In other words, descent direction is a direction that along that line decreases the function.

and α_k is a positive parameter called **step size** that measures the step along the direction p_k.
The direction p_k corresponds in the **gradient descent method** to $-\nabla f(x_k)$, thus

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k)$$

The selection of α_k is crucial task for ensuring convergence to the minimum of a function. A step size that is too small can lead to excessively slow convergence, potentially never reaching the minimum, while a step size that is too large may cause bouncing around the minimum without ever converging to it.



If α_k is chosen with the **backtracking procedure** (Armijo rule) then the algorithm converges to a stationary point of f. The idea is to start from an initial value for α_k, and then reducing it as $\alpha_k = \phi \alpha_k$ with $\phi < 1$ until the following condition is met:

$$f(x_k - \alpha_k \nabla f(x_k)) \leq f(x_k) - \sigma \alpha_k \|\nabla f(x_k)\|^2$$

where σ is typically 0.25 and lies in the range $(0, 0.5)$. The typical value of ϕ is also 0.5.
Since we cannot run infinite iterations (there exists a truncation error), a **stopping criteria** is needed:

- We can use the property of x^*, where $\nabla f(x^*) = 0$, to test whether the approximation x_k is close to the solution within a specified tolerance. The criteria can be:
  - **Absolute criterion**: $\|\nabla f(x_k)\| < \tau_A$
  - **Relative criterion**: $\frac{\|\nabla f(x_k)\|}{\|\nabla f(x_0)\|} < \tau_R$
- We can set a maximum number, k^*, of iterations.
- Additionally, as an heuristic, we stop when the norm of the gradient starts to flatten.

```
input: f, x_0
while stopping criteria holds:
     p_k = - grad(f(x_k))
     a_k = backtrack_procedure
     x_k = x_k + a_k * p_k
     k = k + 1
 output: x_k
```

Initialization. The initial point x_0 influences the local minimum that is found. It is usually chosen randomly within the range of $[-1, 1]$ or $[0, 1]$. Moreover, if $f(x)$ is convex, then every stationary point is a global minimum of $f(x)$ and the choice of x_0 isn't important. Otherwise when $f(x)$ isn't convex, we have to choose an initial point as closest as possible to the *right* stationary point.

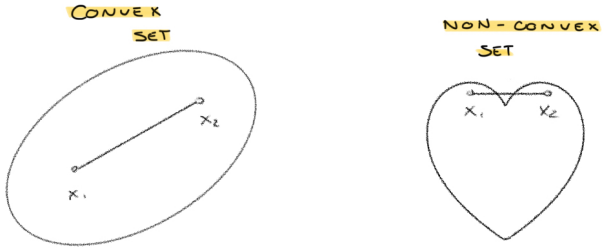
The algorithm introduced doesn't always work well because of the following reasons:

- If the objective function has flat areas or potholes on its loss surface, the procedure might be too slow or lead to a poor solution. Techniques such momentum, that inherit the rate of descent from previous steps, are often able to navigate through local potholes and flat regions. The idea is akin a stone rolling down a hill, gathering speed as it rolls down.
- If the components of the gradient have very different magnitudes, this causes problems for gradient-descent methods.
- If the objective function has a steep region in its loss surface, the descent direction within this area changes quickly. This rapid change increase the likelihood of the divergence.

- The objective function may present non-differentiable points, which can create problems in calculating the gradient.

6.3 Convex functions

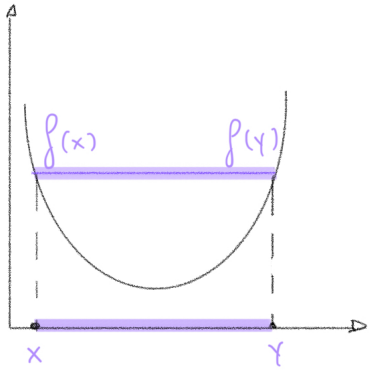
A convex set is a set where, for any two points within the set, the line segment connecting these two points entirely lies within the set.



Definition. *Let $f : \Omega \subset \mathbb{R}^n \to \mathbb{R}$, where Ω is a convex set. The function f is (strictly) **convex** if, $\forall x, y \in \Omega$ and $\forall \theta : 0 \leq \theta \leq 1$, the following inequality holds:*

$$f(\theta x + (1-\theta)y)(<) \leq \theta f(x) + (1-\theta)f(y)$$

In other words, the function of a point lying on the segment connecting x and y is below the segment connecting $f(x)$ and $f(y)$.



6.3.1 Quadratic functions

An example of a strictly convex function is the quadratic function, which take the following form:

$$f(x) = \frac{1}{2}x^T Bx + c^T x + (w)$$

with $b, c \in \mathbb{R}^n$ and $A \in \mathbb{R}^{n \times n}$ symmetric positive definitive.
A typical quadratic function is the least square:

$$\|Ax - b\|^2$$

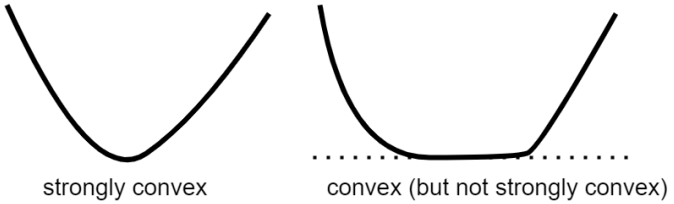
Proof.

$$\begin{aligned}
\frac{1}{2}\|Ax-b\|^2 &= \frac{1}{2}(Ax-b)^T(Ax-b) \\
&= \frac{1}{2}(x^TA^T - b^T)(Ax-b) \\
&= \frac{1}{2}(x^TA^TAx - b^TAx - x^TA^Tb + b^Tb) \\
&= \frac{1}{2}x^T \underbrace{A^TA}_{B} x \underbrace{-b^TA}_{c^T} x + \frac{1}{2}b^Tb
\end{aligned}$$

which is exactly the same as the above form. ■

6.3.2 Properties

1. If f is convex, any point of local minimum is also a global minimum.
2. While a convex function can have multiple local minima, a strictly convex function has only one local minimum, which is also a global minimum.



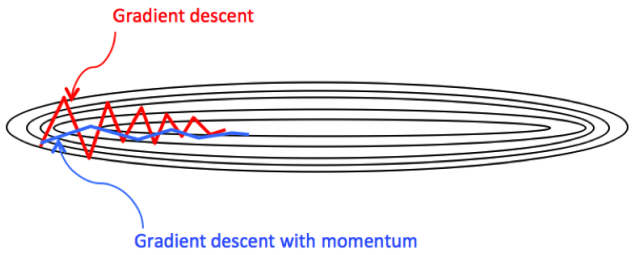
3. If f is convex and differentiable any stationary point is a global minimum for f.

6.4 Variants of gradient descent algorithm

- Gradient descent with **momentum** method improves the convergence of the gradient descent method by memorizing and utilizing, at each step, what happened in the previous iteration. In particular:

  $$x_{k+1} = x_k - \alpha_k \nabla f(x_k) + \beta \Delta x_k$$

  where $\beta \in [0, 1]$ and $\Delta x_k = x_k - x_{k-1}$ is the update obtained at iteration k. This update smooths the gradient updates and, thus, reduces the oscillations. This approach is analogous to a heavy ball in motion, where the momentum term represents the ball's resistance to change directions.



- In machine learning, we typically train models by finding the optimal vector of parameters, denoted as θ, that minimize a loss function $L(\theta)$. We can see this loss functions as the aggregate of individual losses L_n incurred by each of the N data points in the training set. Thus, the loss function is expressed as:

$$L(\theta) = \sum_{n=1}^{N} L_n(\theta)$$

The gradient of such loss function is then computed as follows:

$$\nabla(\theta) = \sum_{n=1}^{N} \nabla L_n(\theta_k)$$

In gradient descent, the optimization is performed by using the full training set and by updating the vectors of parameters according to:

$$\theta_{k+1} = \theta_k - \alpha_k \sum_{n=1}^{N} (\nabla L_n(\theta_k))$$

Evaluating the sum gradient may require expensive evaluations of the gradients from all individual functions L_n. If we consider the term $\sum_{n=1}^{N}(\nabla L_N(\theta_k))$, we can reduce the amount of the computation by taking a sum over a smaller set of L_n.

  - *batch* $\rightarrow$ all L_n functions;
  - *mini-batch* $\rightarrow$ randomly choose a subset of L_n functions. This can be a single L_n or more. Large mini-batches lead to more stable convergence, but the calculations will be more expensive. Small mini-batches are quick to estimate, and the noise in gradient estimation might help to escape from some bad local optima.

The technique is used by **stochastic gradient descent**. It requires more iterations to converge, but with a small enough α_k, it almost surely converges to a local minimum. Why use it? If there are constraints, such as memory limitations.

Moreover, with stochastic gradient descent, we speak about **epoch** and not iterations. An epoch refers to the iterations necessary to "see" all the data.

$$\begin{aligned}
&\nabla L(\theta) = \nabla L_{n_1}(\theta) \quad i = 1,\ n_1 \in \{1 \dots N\} \\
&\nabla L(\theta) = \nabla L_{n_2}(\theta) \quad i = 1,\ n_2 \in \{1 \dots N\} \setminus \{n_1\} \\
&\nabla L(\theta) = \dots
\end{aligned}$$

The mini-batches do not repeat themselves before the entire epoch has been completed.

7 Probability and Statistics

Probability focuses on modeling processes using random variables to capture uncertainty. In contrast, statistics involves analyzing the observed data to deduce the underlying processes that explain those observations.

Definition (State space Ω)**.** *Set of all the possible results of a random experiment.*

Example

Two coins

$$\Omega = \{TT, HT, TH, HH\}$$

Definition (Event space $\mathcal{A}$)**.** *A collection of results and* $\mathcal{A} \subset \Omega$.

Definition (Probability P)**.** *The probability of an event* $\mathcal{A}$ *is a function* $P : A \to [0,1] \in \mathbb{R}$ *that associates at each event* A *a number called probability of* $\mathcal{A}$.

$$P(A) = \frac{\#(A)}{\#(\Omega)}$$

Example

$A \subset \Omega = \{TH, HT\}$

$$P(A) = \frac{2}{4}$$

Definition (Conditional probability)**.** *The conditional probability of an event* B *given the event* A *is defined as:*

$$P(B|A) = \frac{P(A \cap B)}{P(A)}$$

with $P(A) > 0$.

Example

$A = \{\text{The first card is of seed heart}\}$
$B = \{\text{The second card is of seed heart}\}$

$$P(A) = \frac{13}{52} = \frac{1}{4}$$
$$P(B|A) = \frac{12}{51}$$

A and B are events dependent.

Example

Toss a coin three times
$\Omega = \{TTT, TTH, THT, HTT, HHT, HTH, THH, HHH\}$
$A = \{\text{two } T\}$ $B = \{\text{one } H \text{ and one } T\}$
$P(A) = \frac{4}{8}$ $P(B) = \frac{6}{8} = \frac{3}{4}$
$A \cap B = \{TTH, THT, HTT\}$

$$P(A|B) = \frac{\frac{3}{8}}{\frac{3}{4}} = \frac{3}{8}\frac{4}{3} = \frac{1}{2}$$

$$P(B|A) = \frac{\frac{3}{8}}{\frac{1}{2}} = \frac{3}{4}$$

Definition. *Two events* A *and* B *are* <u>independent events</u> *if and only if:*

$$P(A \cap B) = P(A) \cdot P(B)$$

and that means

$$P(A|B) = P(A) \quad P(B|A) = P(B)$$

That can be extended to multiple events. In fact, if

$$P(A_1 \cap A_2 \ldots \cap A_n) = \prod_{i=1}^{n} P(A_i)$$

then $A_1, A_2 \ldots A_n$ *are independent events.*

7.1 Random variables

Definition (Random variable)**.** *A random variable is a function* $X : \Omega \to \mathbb{R}$ *which associates each result* $\omega \in \Omega$ *to a number* $x \in \mathbb{R}$.

Definition. *We refer to* $\mathcal{T}$ *as the* ***target space of*** X *or* ***support of*** X *that is the set of all possible values of a random variable* X.

$$\{x | x = X(\omega), \quad \omega \in \mathbb{R}\}$$

A random variable is:

- **Discrete** if $\mathcal{T}$ is constituted by a <u>countable</u> set of elements.
- **Continue** if $\mathcal{T}$ is an interval or an union of intervals of <u>real</u> numbers.

Example

Toss a coin twice
$\Omega = \{TT, TH, HT, HH\}$
$X(\omega) = \{\text{number of } H\}$
$\omega \in \Omega$
$X(TT) = 0 \; X(TH) = 1, \; X(HT) = 1, \; X(HH) = 2$

$$\mathcal{T} = \{0, 1, 2\}$$

X is discrete r.v.

Example

Roll a die more times until we get 6
$\Omega = \{n_1, n_2 \dots 6\} \; n_i = \{1, \dots 5\}$
$Y = \{\text{number of rolls before getting 6}\}$

$$T = \{1, 2, 3, 4 \dots\}$$

Y is discrete r.v. because it has the same cardinality as integers.

Example

A client enters a bank
$Z = \{\text{time before the arrival of the client}\}$

$$\mathcal{T} = [a, b] \subset (0, \infty)$$

Z is continuous r.v.

Now we will consider particular functions that describe the behavior of random variables, specifically the values that can be obtained from random variables.

7.1.1 Discrete random variables

Definition (Probability mass function)**.** *The* ***probability mass function (PMF)*** *represents the probability that a random variable X takes on a specific value x. Mathematically, this can be expressed as:*

$$f_x : \mathcal{T}_x \to [0, 1]$$

$$\forall x \in \mathcal{T}_x, f_x(x) = P(X = x)$$

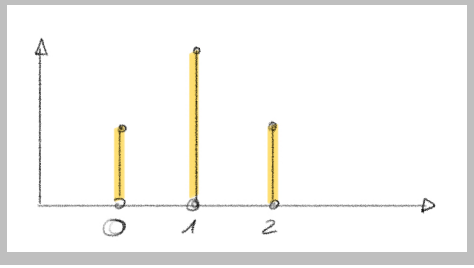
In other words, it represents the probability distribution of a discrete random variable has over its target space.

Example

$\Omega = \{TT, TH, HT, HH\}$
$X = \{\text{number of } H\}$
$\mathcal{T}_x = \{0, 1, 2\}$
$f_x : \{0, 1, 2\} \to [0, 1]$

$$f_x(0) = P(X = 0) = \frac{1}{4}$$
$$f_x(1) = P(X = 1) = \frac{2}{4} = \frac{1}{2}$$
$$f_x(2) = P(X = 2) = \frac{1}{4}$$

This shows that the r.v. has most often the value 1.



Moreover, the PMF has these properties:

1. $f_x(x) \geq 0 \quad \forall x \in \mathcal{T}_x$
2. $\sum_{x \in \mathcal{T}_x} f_x(x) = 1$
3. $A \subset \Omega, P(X = x \in A) = \sum_{x \in A} f_x(x)$

Some examples of PMF are:

- **Uniform distribution**: the r.v takes on all possible values in its target space with equal probability.

  $$f_x(x) = \frac{1}{N}$$

  where $N = \#(\mathcal{T}_x)$.

- **Poisson distribution**: is used to model rare events and it describe the number of events happening in a given unit of time.

  $$f_x(x) = e^{-\lambda} \frac{-\lambda}{x!}$$

  where λ is the mean and the standard deviation.

7.1.2 Continuous random variables

Definition (Probability density function). *X is a continuous r.v. if*

$$f_x : \mathcal{T}_x \to \mathbb{R}$$

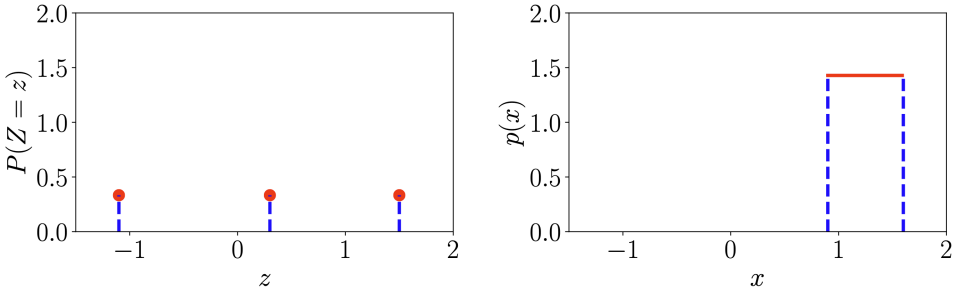
$$P(a \leq X \leq b) = \int_a^b f_x(x)dx$$

where $[a, b]$ is an interval in $\mathbb{R}$.

In contrast to discrete r.v., the probability of a continuous r.v. taking a particular value is zero. In fact:

$$\text{If } a = b \implies \int_a^a f_x(x)dx = 0$$

In other words, the PDF represents the probability of the r.v. falling WITHIN A PARTICULAR RANGE OF VALUES, as opposed to taking on any one value.



(a) Discrete distribution (b) Continuous distribution

Moreover, the PDF has these properties:

1. $f_x(x) \geq 0 \quad \forall x \in \mathcal{T}_x$
2. $\int_a^b f_x(x)dx = 1$

Example

X continuous r.v.
$f_x(x) = 3x^2 \quad 0 \leq x \leq 1 \quad \mathcal{T}_x = [0, 1]$
$P(X \in [0.2, 0.6]) = P(0.2 \leq X \leq 0.6)$?

$$P(0.2 \leq X \leq 0.6) = \int_{0.2}^{0.6} 3x^2 dx = x^3|_{x=0.2}^{0.6} = (0.6^3 - 0.2^3) = 0.208$$

Some examples of PDF are:

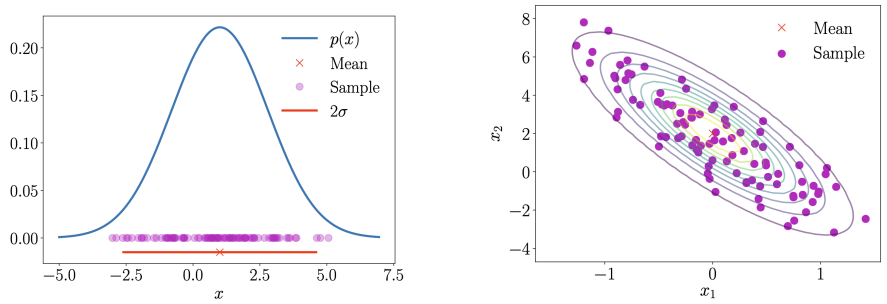
- **Uniform distribution**

$$f_x(x) = \frac{1}{b-a} \quad x \in [a, b] = \mathcal{T}_x$$

- **Normal distribution (Gaussian)**

$$f_x(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}} \quad x \in \mathbb{R}$$

where μ, σ are respectively the mean and the standard deviation.



(a) Univariate (one-dimensional) Gaussian; (b) Multivariate (two-dimensional) Gaus-

The special case of the gaussian with $\mu = 0$ and $\sigma = 1$ is referred as *standard normal distribution.*

7.2 Multiple random variables

Definition. ***Univariate*** *refers to the distribution of a single r.v., while* ***multivariate*** *refers to the distributions of more than one r.v.*

Multivariate Normal distribution.

$$\mathcal{N}(x \mid \mu, \Sigma) = (2\pi)^{-\frac{D}{2}} |\Sigma|^{-\frac{1}{2}} e^{(x-\mu)^T \Sigma^{-1}(x-\mu)}$$

where $x = (x_1, \ldots, x_D)$. When the Gaussian distribution has $\mu = 0$ and $\Sigma = I$, it is referred to as *standard normal distribution.*

Definition (Joint probability). *The target space of each of the* ***joint probability*** *is the Cartesian product of the target spaces of each of the random variables. Mathematically, considering two random variables X and Y and their target spaces, $\mathcal{T}_X$ and $\mathcal{T}_Y$:*

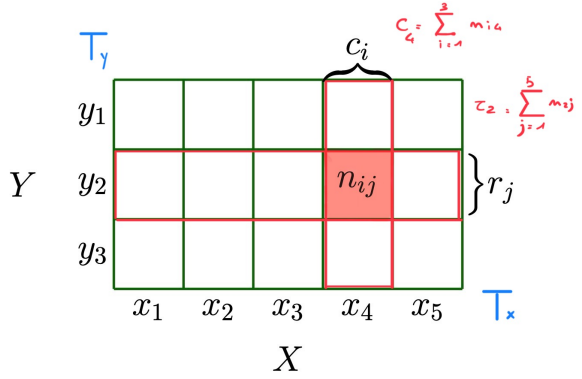
$$\mathcal{T}_{XY} = \mathcal{T}_X \times \mathcal{T}_Y = \{(t_X, t_Y) \mid t_X \in \mathcal{T}_X, t_Y \in \mathcal{T}_Y\}$$

The joint probability is defined as:

$$f_{XY} : \mathcal{T}_{XY} \to [0, 1]$$

$$f_{XY}(x_i, y_j) = P(X = x_i, Y = y_j) = \frac{n_{ij}}{N} \quad x_i, y_j \in \mathcal{T}_{XY}$$

where n_{ij} is the number of the events with state x_i and y_j and N the total number of the events. We can also say that the joint probability is the probability of the intersections of both events, such that $P(X = x_i, Y = y_j) = P(X = x_i \cap Y = y_j)$.



Definition (Marginal probability)**.** *It's the probability of a single event occurring, independent of other event. This can be computed as the sum of the row or column:*

$$f_X(x_i) = P(X = x_i) = \frac{c_i}{N}$$
$$f_Y(y_j) = P(Y = y_i) = \frac{r_i}{N}$$

where c_i and r_i are respectively the ith column and jth row of the probability table.

Definition (Conditional probability)**.** *It's the probability that an event occurs given that another specified event has already happened. This can be computed using the following formulas:*

$$P(Y = y_j \mid X = x_i) = \frac{P(X = x_i \cap Y = y_j)}{P(X = x_i)} = \frac{n_{ij}}{c_i}$$
$$P(X = x_i \mid Y = y_j) = \frac{P(Y = y_i \cap X = x_j)}{P(Y = y_i)} = \frac{n_{ij}}{r_j}$$

Given the definitions of the marginal and conditional probability for ours r.v. we can now present two rules in probability theory:

- **Sum rule**:

$$f_X(x_i) = \begin{cases} \sum_{y_j \in \mathcal{T}_y} f_{XY}(x_i, y_j) & \text{if } y \text{ is discrete} \\ \int_{\mathcal{T}_Y} f_{XY}(x_i, y_j) dy_j & \text{if } y \text{ is continuous} \end{cases}$$

- **Product rule**: the joint distribution of two random variables can be factorized as conditional distribution and marginal distribution

$$f_{XY}(x_i, y_j) = P(Y = y_j \mid X = x_i) f_X(x_i)$$

or

$$f_{XY}(x_i, y_j) = P(X = x_j \mid Y = y_i) f_Y(y_j)$$

Theorem (Bayes's theorem)**.** *The bayes theorem states how to update the prior with new information given by the likelihood. In particular:*

$$\underbrace{P(x \mid y)}_{posterior} = \frac{\overbrace{P(y \mid x)}^{likelihood}\overbrace{P(x)}^{prior}}{\underbrace{P(y)}_{evidence}}$$

- ***Prior**: encapsulates our subjective prior knowledge of the unobserved variable x before observing any data. We can choose any prior that makes sense.*
- ***Likelihood**: it is the probability of the data y if we were to know the latent variable x.*
- ***Posterior**: what we are interest in, i.e, what we know about x after having observed y.*
- ***Evidence**: acts like a normalizing constant.*

It's possible to focus on some statistic of the posterior, such as the maximum of the posterior.

$$\arg\max_x P(x \mid y) \underset{\text{Bayes}}{=} \frac{P(y \mid x)P(x)}{\cancel{P(y)}}$$

7.3 Summary statistics

The statistic of a r.v. is a deterministic function of that r.v.. The summary statistic provide one useful view of how a r.v. behave and provide numbers that summarize and characterize it.

Definition (Expected Value). *The expected value of a function $g : \mathbb{R} \to \mathbb{R}$ of a univariate r.v. X is given by*

$$\mathbb{E}_X[g(x)] = \begin{cases} \int_{\mathcal{T}_X} g(x) f_X(x) & \text{if } X \text{ is continuous} \\ \sum_{x \in \mathcal{T}_X} g(x) f_X(x) & \text{if } X \text{ is discrete} \end{cases}$$

For multivariate r.v. $X = [X_1, \dots, X_D]$, the expected value is a vector of expected values of the respective univariate r.v.:

$$\mathbb{E}_X[g(x)] = \begin{bmatrix} \mathbb{E}_{X_1}[g(x_1)] \\ \vdots \\ \mathbb{E}_{X_D}[g(x_D)] \end{bmatrix} \in \mathbb{R}^D$$

The definition of the mean is a special case of the expected value, obtained by choosing g to be the identity function $g(x) = x$.

Definition (Mean). *The mean of a multivariate r.v. $X = [X_1, \dots, X_D]$ is an average and is defined as*

$$\mathbb{E}_X[g(x)] = \begin{bmatrix} \mathbb{E}_{X_1}[x_1] \\ \vdots \\ \mathbb{E}_{X_D}[x_D] \end{bmatrix} \in \mathbb{R}^D$$

where each one corresponds to

$$\begin{cases} \int_{\mathcal{T}_X} x f_X(x) & \text{if } X \text{ is continuous} \\ \sum_{x \in \mathcal{T}_X} x f_X(x) & \text{if } X \text{ is discrete} \end{cases}$$

The mean is also called the expected value because it represents the most probable value.

Example

X continuous r.v.
$f_x(x) = 3x^2 \quad 0 \le x \le 1$
mean of $f_x(x)$?

$$\mathbb{E}_x[x] = \int_0^1 x f_x(x) dx = \int_0^1 3x^3 dx = \frac{3}{4}x^4 \,|_{x=0}^1 = \frac{3}{4}$$

Definition (Covariance). *The Covariance between two univariate random variable $X, Y \in \mathbb{R}$ is given by the expected product of their deviations from their respective means:*

$$Cov_{X,Y}[x, y] = \mathbb{E}_{X,Y}[(x - \mathbb{E}_X[x])(y - \mathbb{E}_Y[y])]$$

When dealing with two multivariate r.v., $X = (X_1, \dots, X_D)$ and $Y = (Y_1, \dots, Y_E)$, the covariance between X and Y is defined as follows:

$$Cov_{X,Y}[x, y] = \begin{bmatrix} Cov[x_1, y_1] & \dots & Cov[x_1, y_E] \\ \vdots & \ddots & \vdots \\ Cov[x_D, y_1] & \dots & Cov[x_D, y_E] \end{bmatrix} \in \mathbb{R}^{D \times E}$$

*The $D \times E$ matrix is called the **cross-covariance matrix** of the two multivariate r.v. X and Y.*

Definition (Variance). *The covariance of a variable with itself $Cov_{X,X}[x, x]$ is called **variance** and is defined as follows:*

$$\begin{aligned} \mathbb{V}_X[x] &= Cov_X[x, x] \\ &= \begin{bmatrix} Cov[x_1, x_1] & \dots & Cov[x_1, x_D] \\ \vdots & \ddots & \vdots \\ Cov[x_D, x_1] & \dots & Cov[x_D, x_D] \end{bmatrix} \end{aligned}$$

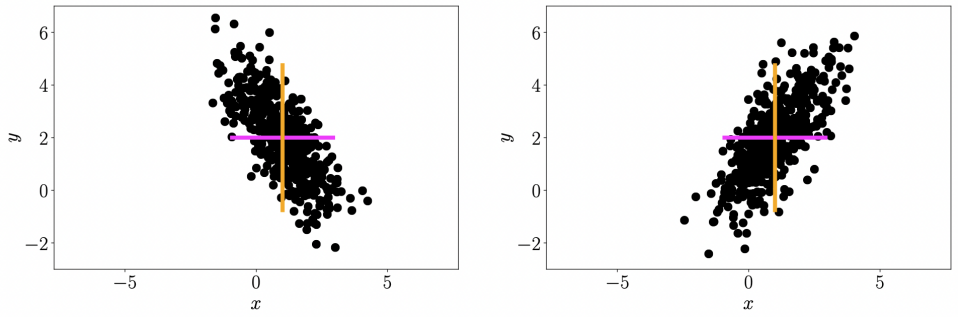
*The $D \times D$ matrix is called the **covariance matrix** of the multivariate r.v. X.*

The covariance indicates how two r.v. are related in terms of their dimensions. However, since covariance lacks an upper bound, it doesn't provide a clear indication how much they are related. To address this, a normalized version of covariance, which constraints the range between -1 and 1, offer a more convenient statistic to measure the relationship between two r.v.

Definition (Correlation). *The correlation between two r.v. X, Y is given by*

$$corr_{X,Y}[X, Y] = \frac{Cov_{X,Y}[x, y]}{\sqrt{\mathbb{V}_X[x]\mathbb{V}_Y[y]}} \in [-1, 1]$$

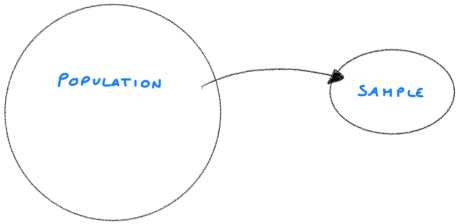
Positive correlation means that when x grows, the y is also expected to grow. Negative correlation means that as x increases then y decreases



(a) x and y are negatively correlated.

(b) x and y are positively correlated.

Empirical statistics The definitions above are called population mean and population covariance, as it refers to the true statistics for the population.



From a dataset that represents a sample of the population, we observe it and infer some properties of the population:

- **Empirical mean**:

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

- **Empirical variance**:

$$\mathbb{V}_X[x] = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})$$

- **Empirical standard definition**:

$$\sigma = \sqrt{\mathbb{V}_X[x]}$$

7.4 Algebra of random variables

Considering two r.v. $X = (X_1, \ldots, X_D)$ and $Y = (Y_1, \ldots, Y_D)$, then:

$$\begin{aligned}
\mathbb{E}_{X,Y}[x+y] &= \mathbb{E}_X[x] + \mathbb{E}_Y[y] \\
\mathbb{E}_{X,Y}[x-y] &= \mathbb{E}_X[x] - \mathbb{E}_Y[y] \\
\mathbb{V}_{X,Y}[x+y] &= \mathbb{V}_X[x] + \mathbb{V}_Y[y] + \mathrm{Cov}_{X,Y}[x,y] + \mathrm{Cov}_{Y,X}[y,x] \\
\mathbb{V}_{X,Y}[x-y] &= \mathbb{V}_X[x] + \mathbb{V}_Y[y] - \mathrm{Cov}_{X,Y}[x,y] - \mathrm{Cov}_{Y,X}[y,x]
\end{aligned}$$

Definition (Independence)**.** *Two r.v., X and Y, are statistically independent if the joint probability of X and Y can be written as the product of their marginal probabilities:*

$$f_{X,Y}[x,y] = f_X[x] f_Y[y]$$

Intuitively, this means that the two r.v. don't influence each other.

If X, Y are independent, then:

$$\begin{aligned}
P(Y = y \mid X = x) &= P(Y = y) \\
P(X = x \mid Y = y) &= P(X = x) \\
\mathbb{V}_{X,Y}[x+y] &= \mathbb{V}_X[x] + \mathbb{V}_Y[y] \\
\mathrm{Cov}_{X,Y}[x,y] &= 0
\end{aligned}$$

The viceversa of the last point doesn't hold.

Definition (Conditional Independence)**.** *Two r.v., X and Y, are conditionally independent given Z if and only if*

$$P(X = x \cap Y = y \mid Z = z) = P(X = x \mid Z = z) P(Y = y \mid Z = z)$$

Geometry. r.v. can be considered vectors in a vector space of probability distributions, and we can define inner product to obtain geometric properties of it. Recall the definition of inner products $\langle \cdot, \cdot \rangle : (x, y) \to \mathbb{R}$, so for zero mean r.v. X and Y

$$\langle X, Y \rangle = \mathrm{Cov}_{X,Y}[x,y] \in \mathbb{R}^+$$

The length of a r.v. is the standard deviation:

$$\|X\| = \sqrt{\mathrm{Cov}_{X,Y}[x,x]} = \sqrt{\mathbb{V}[x]} = \sigma[x]$$

The longer the r.v, the more uncertain it is; and a r.v with length 0 is deterministic. We can interpret the angle θ between two r.v., X and Y, as their correlation:

$$\cos\theta = \frac{\langle X, Y \rangle}{\|X\|\|Y\|} = \frac{\mathrm{Cov}_{X,Y}[x,y]}{\sqrt{\mathbb{V}_X[x]\mathbb{V}_Y[y]}}$$

Recall that two vectors are orthoganal if and only if:

$$X \perp Y \iff \langle X, Y \rangle = 0$$

Therefore, they are orthogonal if and only if $\mathrm{Cov}_{X,Y}[x,y] = 0$.

Gaussian linearity. For two independent r.v., X, Y, we have:

$$\begin{aligned}
f_{X,Y}[x+y] &= \mathcal{N}(\mu_x + \mu_y, \Sigma_x + \Sigma_y) \\
f_{X,Y}[\alpha x + \beta y] &= \mathcal{N}(\alpha\mu_x + \beta\mu_y, \alpha^2\Sigma_x + \beta^2\Sigma_y)
\end{aligned}$$

where $\alpha, \beta \in \mathbb{R}$. This means that in both cases, the result will be a Gaussian distribution.

8 Machine Learning Problems

In machine learning, our main goal, is to create models, or predictors, that make good predictions on unseen data. The concept of a model can be seen in two ways:

- **as function (deterministic).** A predictor is a function:

  $$f : \mathbb{R}^D \to \mathbb{R}$$

  Here, the input vector x is a D-dimensional (that has D features), and the function f, when applied to this vector, returns a real number. We will focus on linear functions, which can be represented as:

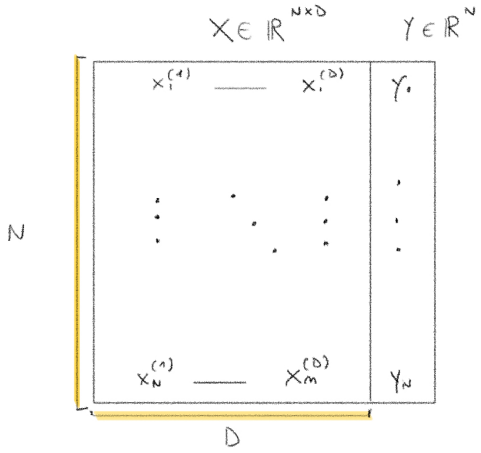
  $$f(x) = \theta^T x$$

  where $\theta = [\theta_0, \theta_1, \ldots, \theta_D]$ and $x = [1, x^{(1)}, \ldots, x^{(D)}]$ (we have concatenated an unit feature $x^{(0)} = 1$ to x, which allows us to incorporate the bias term into the vector θ).
  The parameters of the function are represented by θ.

- **as multivariate probability distribution (probabilistic).** In this case, the parameters correspond to those of the distribution.

In order to obtain good predictions from a model, it's essential to adjust its parameters using training data (learning). There are two main approaches to this parameter adjustment: for non-probabilistic models, we use **empirical risk minimization**, and for probabilistic models, we utilize the method of **maximum likelihood** estimation.

8.1 Empirical Risk Minimization



Given a dataset consisting of N examples $x_n \in \mathbb{R}^D$ and corresponding scalar labels $y_n \in \mathbb{R}$, we consider a supervised learning scenario. In this setting, we work with pairs $(x_1, y_1), \ldots, (x_N, y_N)$. The objective is to identify the optimal parameters θ^* of a predictor $f(\cdot, \theta) : \mathbb{R}^D \to \mathbb{R}$, such that the model fits well the data. In other words, we aim for

$$f(x_n, \theta^*) = \hat{y} \approx y_n \quad \forall n = 1, \ldots, N$$

To define what it means to fit the data well, we need to specify a **loss function** $\ell(y_n, \hat{y}_n)$, which take the ground truth label y_n and the prediction $\hat{y}_n$ as input and produces a non-negative number representing how much error the model has been made on the particular prediction.
The optimal parameter vector θ^* is given by minimizing the average loss of the set of N training samples that is given by **empirical risk**:

$$R_{\text{emp}(f,X,y)} = \frac{1}{N} \sum_{n=1}^{N} \ell(y_n, \hat{y}_n)$$

$$\theta^* = \min_{\theta \in \mathbb{R}^D} \frac{1}{N} \sum_{n=1}^{N} \ell(y_n, \hat{y}_n)$$

Example

An example of a loss is the least-squares $\ell(y_n, \hat{y}_n) = (y_n - \hat{y}_n)$, thus:

$$\theta^* = \min_{\theta \in \mathbb{R}^D} \frac{1}{N} \sum_{n=1}^{N} (y_n - f(x_n, \theta))^2$$

where we substituted the predictor $\hat{y}_n = f(x_n, \theta)$. By using the linear predictor $f(x_n\theta) = \theta^T x_n$ we obtain:

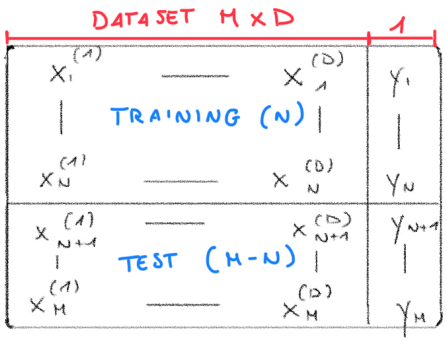
$$\theta^* = \min_{\theta \in \mathbb{R}^D} \frac{1}{N} \sum_{n=1}^{N} (y_n - \theta^T x_n)^2$$

That is equivalently expressed in matrix form:

$$\theta^* = \min_{\theta \in \mathbb{R}^D} \frac{1}{N} \|y - X\theta\|^2$$

This is known as the *least-squares problem.*

However, we aren't interested in a predictor that only perform well on the training data. Instead, we seek a predictor that performs well on unseen data , i.e. the predictor generalizes well.



Given a dataset constituted of M examples $x_m \in \mathbb{R}^D$ and corresponding scalar labels y_m, we divide the dataset into two parts: a training set with N examples and a **test set** with the remaining $M - N$ examples. We simulate the unseen data by using this test set.
Now we are interested in finding a predictor f (with parameters fixed) that minimized the **expected risk**:

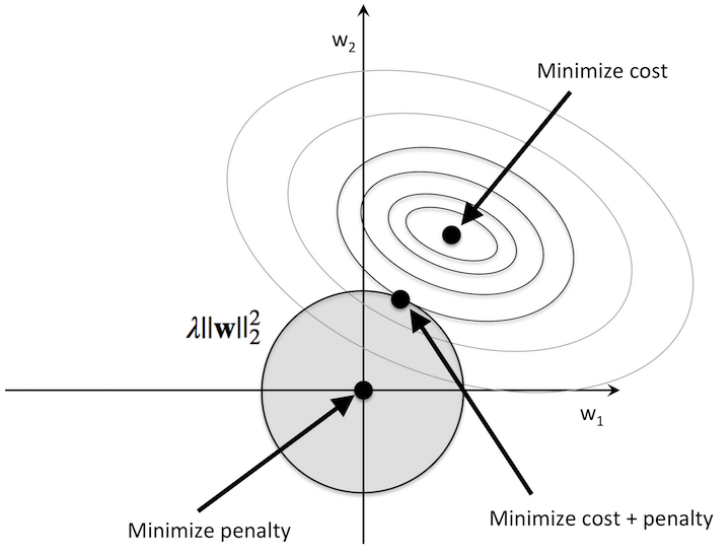
$$R_{\text{true}}(f) = \mathbb{E}_{x,y}[\ell(y, f(x))]$$

Here, minimizing the expected risk means finding the predictor that has the lowest loss when evaluated on the entire population, which in our scenario is represented by the test set.

Regularization. When the test risk is much larger than the training risk, this is an indication of **overfitting**, i.e. the predictor fits too closely to the training data and doesn't generalize well to new data. This issue often arises when utilizing overly complex models. To address this, we introduce a penalty term involving θ, which serves to reduce the model's complexity:

$$\theta^* = \min_{\theta \in \mathbb{R}^D} \frac{1}{N} \|y - X\theta\|^2 + \lambda \|\theta\|^2$$

The additional term $\|\theta\|^2$ is called *regularizer*, and the hyperparameter λ is the *regularization parameter*. This hyperparameter trades off minimizing the loss on the training set and the magnitude of the parameter θ. A sign of overfitting is when the values of the parameters become large. By applying the penalty term, we essentially restrict the vector θ to remain closer to the origin.



8.2 Parameter estimation of a probability distribution

In a supervise learning setting, we have a dataset consisting of N pairs $(x_1, y_1) \ldots, (y_N, y_N)$, where each $x_n \in \mathbb{R}^{\mathbb{D}}$ and $y_n \in \mathbb{R}$ (for $n = 1, \ldots, N$). We assume these pairs are independently identically distributed.
Each vector x_n is drawn from an unknown probability distribution $x_n \sim P(x)$. The corresponding label y_n is considered as an output from a conditional probability distribution of labels given the examples for the particular parameter setting θ, $y_n \sim P_\theta(y \mid x_n)$.

8.2.1 Maximum Likelihood Estimation

With this framework, learning a model can be thought as finding the parameters of a distribution $P_\theta(y \mid x)$ (the likelihood) that maximized the probability of observing y, given x. Therefore, we must solve the optimization problem:

$$\theta^* = \arg\max_\theta P_\theta(y \mid x)$$

which is called **Maximum Likelihood Estimation (MLE)**, because the parameters θ^* are chosen such that they maximize the likelihood.
Given the assumption of independence, we can factorized the whole dataset into a product o the likelihoods of each individual example:

$$P_\theta(x, y) = \prod_{n=1}^{N} P_\theta(y_n \mid x_n)$$

Thus, the MLE can be reformulated as:

$$\theta^* = \arg\max_\theta \prod_{n=1}^{N} P_\theta(y_n \mid x_n)$$

Here, $P_\theta(y_n \mid x_n)$ represents a particular distribution, and all of these distribution are the same that share the same parameters.
Since the logarithm function is monotonic, applying it to the optimization problem does not alter its solution. Furthermore, since for any function $f(x)$, $\arg\max_x f(x) = \arg\min_x -f(x)$ can be restated as:

$$\theta^* = \arg\min_\theta -\log \prod_{i=1}^{N} P_\theta(y_n \mid x_n)$$

which is the classical formulation of an MLE problem.

Gaussian distribution. If we specify a Gaussian likelihood for each example pair x_n, y_n as

$$P_\theta(y_n \mid x_n) = \mathcal{N}(y_n \mid f_\theta(x_n), \sigma^2)$$

the negative likelihood can be rewritten as

$$\begin{aligned}
\theta^* &= -\sum_{n=1}^{N} \log P_\theta(y_n \mid x_n) \\
&= -\sum_{n=1}^{N} \log \mathcal{N}(f_\theta(x_n), \sigma^2) \\
&= -\sum_{n=1}^{N} \log \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_n - f_\theta(x_n))^2}{2\pi^2}} \\
&= -\sum_{n=1}^{N} \log e^{-\frac{(y - f_\theta(x))^2}{2\pi^2}} - \sum_{n=1}^{N} \log \frac{1}{\sqrt{2\pi\sigma^2}} \\
&= \frac{1}{2\sigma^2} \sum_{n=1}^{N} (y - f(x_n))^2 - \sum_{n=1}^{N} \log \frac{1}{\sqrt{2\pi\sigma^2}}
\end{aligned}$$

Since the second term is constant in minimizing the MLE, MLE with Gaussian likelihood becomes

$$\theta^* = \arg\min_\theta \sum_{n=1}^{N} \frac{1}{2} (y_n - f_\theta(x_n))^2$$

which can be reformulated as a Least Squares problem:

$$\theta^* = \arg\min_\theta \frac{1}{2} \|f_\theta(x) - y\|^2$$

where $y = [y_1, \ldots, y_N]$, while $f_\theta(x) = [f_\theta(x_1), \ldots, f_\theta(x_N)]$.

8.2.2 Maximum A Posteriori Estimation

Maximum likelihood estimation may suffer from overfitting. A different approach is to stop using MLE. The idea is to reverse the problem and, instead of searching the parameters θ such that the probability of observing the outcomes y given the data x is maximized, i.e. maximizing $P_\theta(y \mid x)$, as in MLE, try to maximize the probability of the parameters θ given the observed data is (x, y). Mathematically, we want to solve:

$$\theta^* = \arg\max_\theta P(\theta \mid x, y)$$

Since $P(\theta \mid x, y)$ is the *posterior distribution*, this method is usually referred to as a **Maximum A Posteriori (MAP)**. we can express it in terms of the likelihood $P(y \mid x, \theta)$ and the prior $P(\theta)$, as a consequence of Bayes Theorem. Thus, MAP can be rewritten as

$$\theta^* = \arg\max_\theta P(y \mid x, \theta) P(\theta)$$

Just as with MLE, we can change to minimum point estimation by changing the sign an applying the logarithm:

$$\theta^* = \arg\min_\theta -\log P(y \mid x, \theta) - \log P(\theta)$$

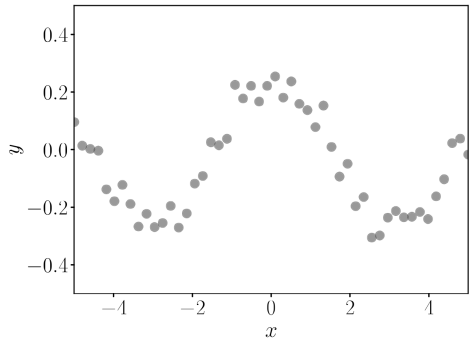
If we assume that $P(\theta) = \mathcal{N}(0, \sigma^2)$ is a Gaussian distribution with zero mean. Thus,

$$-\log P(\theta) = \frac{1}{2\sigma^2} \|\theta\|^2$$

That introduces an additional term that biases the resulting parameters to be close to the origin.

9 Polynomial Linear Regression

Given a dataset consisting of N pairs $(x_n, y_n)_{n=1,\dots,N}$ with inputs $x_n \in \mathbb{R}^D$ and function values $y_n = f(x_n) + \epsilon$, where ϵ is a i.i.d. r.v. that models the noise following a zero-mean Gaussian distribution $\epsilon \sim \mathcal{N}(0, \sigma^2)$. As a result, y_n follows the same distribution due to this noise.



Our goal is to find a function that is similar to the unknown function f that generated the data. If we consider a standard linear regression, represented as $f(x_n) = x_n^T\theta$, we can only fit straight lines to data. However, if we consider the non linear transformation $f(x_n) = \phi^T(x_n)\theta$, which is still linear in terms of θ, we get the corresponding linear model $y_i = \phi^T(x_n)\theta + \epsilon_i$ where $\phi : \mathbb{R}^D \to \mathbb{R}^K$ and $\phi_k : \mathbb{R}^D \to \mathbb{R}$:

$$\phi(x) = \begin{bmatrix} \phi_0(x) \\ \phi_1(x) \\ \vdots \\ \phi_{K-1}(x) \end{bmatrix} \in \mathbb{R}^K$$

Example

For a second polynomial the transformation $\phi(x)$ is defined as

$$\phi = \begin{bmatrix} 1 \\ x \\ x^2 \end{bmatrix}$$

The original one-dimensional space can be "lifted" into a K-dimensional feature space. If we choose all monomials x^k for $k = 0, \dots, K-1$ we are able to model polynomials $\leq K-1$ within the framework of linear regression. A polynomial of degree $K-1$ is

$$\begin{aligned} f(x_n) &= \phi^T(x_n)\theta \\ &= \sum_{n=0}^{K-1} \theta_n x^n \end{aligned}$$

Considering all N training inputs, we can define the *feature matrix* as

$$\Phi = \begin{bmatrix} \phi^T(x_1) \\ \vdots \\ \phi^T(x_N) \end{bmatrix} = \begin{bmatrix} \phi_0^T(x_1) & \dots & \phi_{K-1}(x_1) \\ \phi_0^T(x_2) & \dots & \phi_{K-1}(x_2) \\ \vdots & & \vdots \\ \phi_0^T(x_N) & \dots & \phi_{K-1}(x_N) \end{bmatrix} \in \mathbb{R}^{N \times K}$$

where $\Phi_{ij} = \phi_j(x_i)$ and $\phi_j : \mathbb{R}^D \to \mathbb{R}$.

Example

For a second-order polynomial and N training points $x_n \in \mathbb{R}$, $n = 1, \dots, N$, the feature matrix is

$$\Phi = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \vdots \\ 1 & x_N & x_N^2 \end{bmatrix}$$

With the feature matrix Φ defined, the negative log-likelihood for the linear regression model can be written as

$$-\log P_\theta(y|x) = \frac{1}{2\sigma^2}\|y - \Phi\theta\|^2$$

By treating the $\frac{1}{2\sigma^2}$ as a constant, the minimization problem can be thought as least square problem:

$$\theta^* = \arg\min_\theta \|y - \Phi\sigma\|^2$$

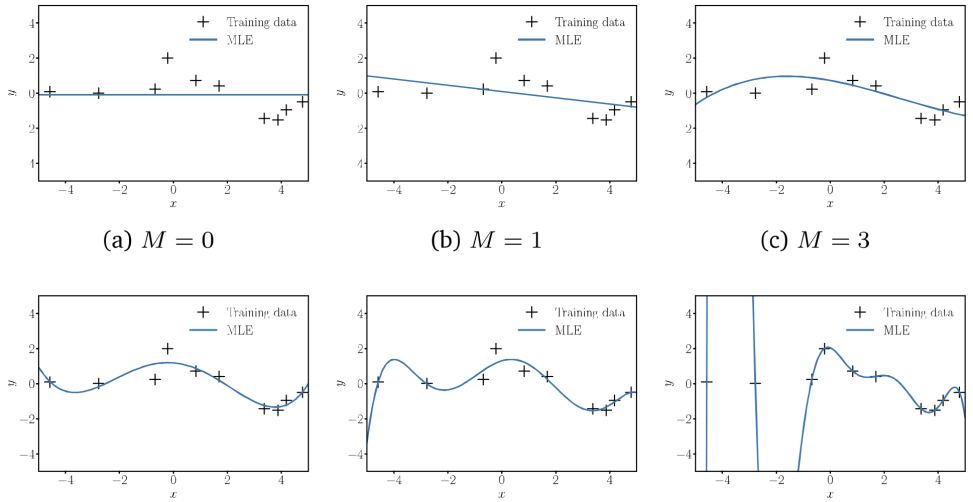
This least squares problem can be solved using the *normal equations* or SGD:

$$\begin{aligned} \min_x \|Ax - b\|^2 \\ A^TAx - A^Tb = 0 \\ A^TAx = A^Tb \\ x = (A^TA)^{-1}A^Tb \end{aligned}$$

Applying this to our minimization problem:

$$\begin{aligned} \Phi^T\Phi\theta = \Phi^Ty \\ \theta^* = (\Phi^T\Phi)^{-1}\Phi^Ty \end{aligned}$$

To get a better fit for our predictor, represented as $f(x) = \sum_{n=0}^{K-1} \theta^* x^n$, on the training data, we can vary the degree of the polynomial.



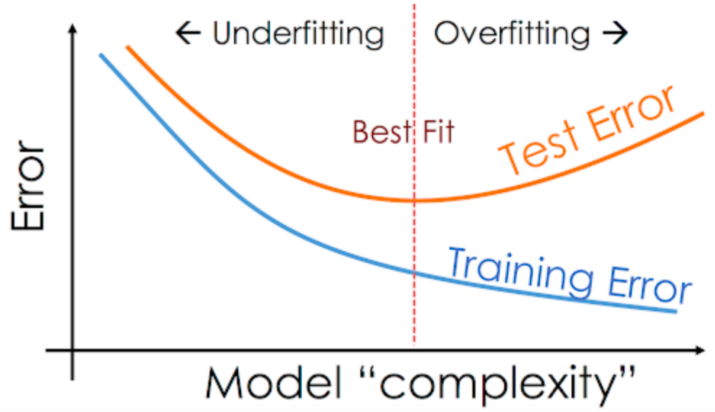
(a) $M = 0$ (b) $M = 1$ (c) $M = 3$

(d) $M = 4$ (e) $M = 6$ (f) $M = 9$

However, our goal is to have a predictor that generalizes well. Therefore, we plot the loss on both the training and test sets while varying the polynomial degree.
A more robust metric for assessing the model's quality is by computing the *root mean square error (RMSE)*, defined as:

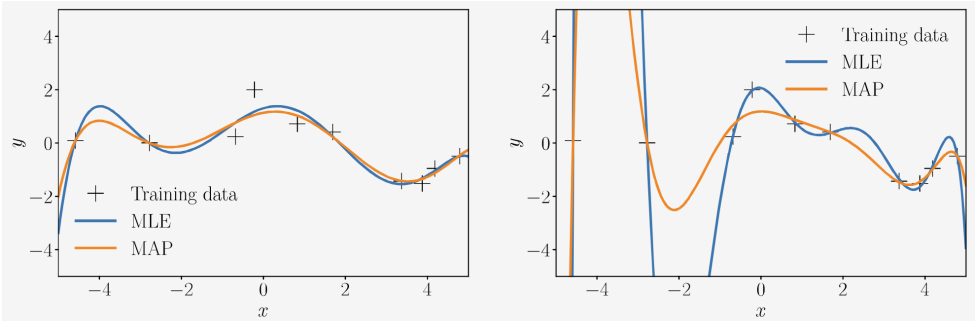
$$\sqrt{\|y - \Phi\theta\|^2/N} = \sqrt{\frac{1}{N}\sum_{n=1}^{N}(y_n - \phi^T(x_n)\theta)^2}$$

which allows us to compare errors of dataset with different sizes and has the same unit measure of the observed values y_n.
When observing that RMSE decreases on the training set but increases on the test, it is indicative of overfitting. Again, we can plot the RMSE while varying polynomial degree and opt for the higher one just before the overfitting divergence happens.



Another way to mitigate overfitting is through the use of regularization, transitioning from MLE to MAP:

$$\theta^* = \arg\min_{\theta} \|y - \Phi\theta\|^2 + \lambda\|\theta\|^2$$



(a) Polynomials of degree 6. (b) Polynomials of degree 8.