Elements of Linear Algebra

Danilo Comminiello

Neural Networks 2023/2024

September 28, 2023

Sapienza
Università di Roma

Table of contents

1 Basic Definitions

Introduction to Linear Algebra

Basic Definitions

Linear algebra for Neural Networks

In order to address the course topics, it is very important to be comfortable with linear algebra.

Working with vectors and matrices must *not* represent an obstacle.

You need to be able to mostly use matrix and vector product notation *quickly* and *easily*.

The best tip to learn linear algebra is to do a lot of practice problems.

Programming tools, like Python, can help you to learn.

Basic concepts

- **Linear algebra**: provides the main rules for manipulating vectors and matrices.
- **Matrix**: is defined as a table of numbers.
- **Vector**: is a particular case of the matrix.
- **Tensor**: is a generalization of a matrix.

Advantages of linear algebra

- Compact notation.
- Intuitive geometric representation.
- Convenient set of operations.
- Suitable data structures for signal manipulations.
- Widely used in modern textbooks and papers.
- Easy equation-to-code translation with many programming tools.

Linear algebra in algorithm development

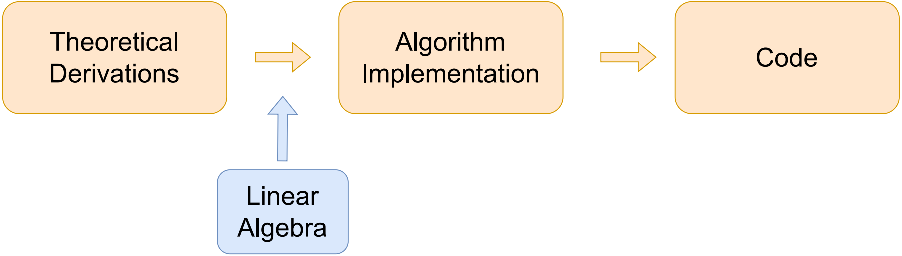


Figure 1: Linear algebra is fundamental to easily encode theory in an algorithm, whose form allows a direct implementation of math operations.

Scalars

A **scalar** is a physical quantity that is described by any real number.

It is usually denoted by a lowercase letter, e.g., x:

$$x_1 = 1, \quad x_2 = 0.3, \quad x_3 = -2, \; \ldots$$

With reference to signals, a scalar may represent for example:

- the amplitude value of a signal sample,
- the color for 1 pixel of an image.

A **vector** is an ordered linear arrangement of a set of scalars. It is usually denoted with a lowercase boldface letter, e.g., $\mathbf{x}$:

$$\mathbf{x} = [x_1 \quad x_2 \quad x_3] = [1 \quad 0.3 \quad -2] \,.$$

Vectors can be used to identify the path from origin to a location P in an N-dimensional space.

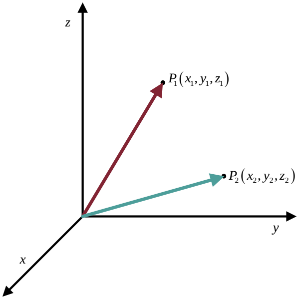


Figure 2: A geometrical representation of vectors.

Vectors II

With reference to signals, a vector may represent:

- 1-dimensional digital signals (e.g., audio signals, brain signals, time-series) which are composed of an ordered sequence of samples (i.e., a sequence of scalars).

As a common notation, 1D signals are usually stored in column vectors.

$\mathbf{x}^T =$

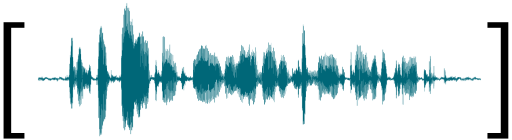


Figure 3: Vector is a typical data structure for 1D signals, like audio waveforms.

Matrices I

A **matrix** consists of set of ordered elements arranged in a number N of rows and M columns.

It is usually denoted with a bold capital letter, e.g., $\mathbf{X}$.

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \dots & \mathbf{x}_M \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1M} \\ x_{21} & x_{22} & \cdots & x_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N1} & \cdots & x_{NM} \end{bmatrix}$$

A matrix can be seen as a vertical stacking of row vectors or as a horizontal arrangement of column vectors.

Matrices II

With reference to signals, matrices may represent:

- 2-dimensional digital signals (e.g., images),
- collections of 1D signals (e.g., multichannel audio, multisensor signals).

$\mathbf{X} =$

Figure 4: Matrix is a typical data structure for 2D signals.

Tensors I

A **tensor** is a linear arrangement of a set of K matrices.

While matrices are functions of two indices for rows and columns, tensors are functions of three or more indices.

It is usually denoted with a capital letter of a special font face, e.g., X.

$$\mathsf{X} = \left\{ \mathbf{X}_1 \quad \mathbf{X}_2 \quad \ldots \quad \mathbf{X}_K \right\}$$

A tensor is a generalization of a matrix.

It can be seen as a stacking of horizontal, vertical or lateral slices (e.g., matrices or higher-dimensional tensors).

Tensors II

With reference to signals, tensors may represent:

- 3-dimensional digital signals (e.g., video signals, time-frequency representation of EEG signals),
- or even higher-dimensional signals (e.g., MR brain imaging).

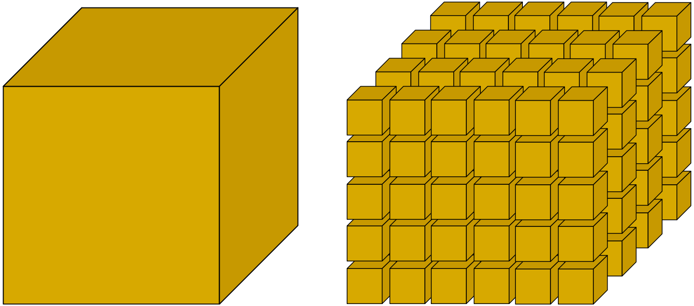


Figure 5: Typical examples of 3D tensor and higher-dimensional tensor.

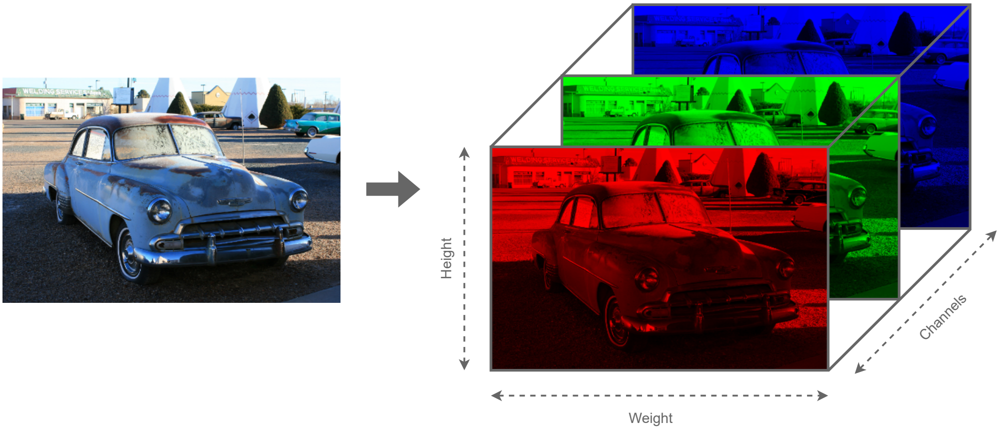


Figure 6: A color image can be stored in a 3D tensor X $= \{\ \mathbf{X}_{\text{R}}\quad \mathbf{X}_{\text{G}}\quad \mathbf{X}_{\text{B}}\ \}$, in which each matrix $\mathbf{X}_i$, with $i = \{\text{R, G, B}\}$, represents an image channel.

Additions and multiplications with matrices

Addition of matrices: Given two matrices $\mathbf{A}, \mathbf{B} \in (\mathbb{R}, \mathbb{C})^{N\times M}$ with the same size, their sum $\mathbf{C} \in (\mathbb{R}, \mathbb{C})^{N\times M} = \mathbf{A} + \mathbf{B}$ is defined by the sum of individual entries with the same index

$$c_{ij} = a_{ij} + b_{ij}, \quad i = 1,\ 2,\ \ldots,\ N;\ \ j = 1,\ 2,\ ...,\ M.$$

Matrices multiplication: Given two matrices $\mathbf{A} \in (\mathbb{R}, \mathbb{C})^{N\times P}$ and $\mathbf{B} \in (\mathbb{R}, \mathbb{C})^{P\times M}$, their product $\mathbf{C} \in (\mathbb{R}, \mathbb{C})^{N\times M} = \mathbf{A} \cdot \mathbf{B}$, is defined as

$$c_{ij} = \sum_{k=1}^{P} a_{ik} b_{kj}, \quad i = 1,\ 2,\ \ldots,\ N;\ \ j = 1,\ 2,\ ...,\ M.$$

Multiplications in Python

Remark

In Python it is possible to perform multiplications and divisions using the following commands:

- `A@B`, performs multiplication of matrices;
- `A*B`, performs element-wise multiplication of matrices;
- `A/B`, performs element-wise division of matrices;
- `A**p`, performs element-wise exponentiation of order p.

Kroneker product

The **Kronecker product** between two matrices $\mathbf{A} \in (\mathbb{R}, \mathbb{C})^{P \times Q}$ and $\mathbf{B} \in (\mathbb{R}, \mathbb{C})^{N \times M}$, usually denoted as $\mathbf{A} \otimes \mathbf{B}$, is defined as

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & \cdots & a_{1Q}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{P1}\mathbf{B} & \cdots & a_{PQ}\mathbf{B} \end{bmatrix} \in (\mathbb{R}, \mathbb{C})^{PN \times QM}.$$

Remark

The Kronecker product of matrices corresponds to the *abstract tensor product* of linear maps. Specifically, if matrices $\mathbf{A}$ and $\mathbf{B}$ represent the linear applications $T_1 : X_1 \to Y_1$ and $T_2 : X_2 \to Y_2$, then the matrix $\mathbf{A} \otimes \mathbf{B}$ represents the tensor product between the two maps $X_1 \otimes X_2 \to Y_1 \otimes Y_2$.

In Python, $\mathbf{A} \otimes \mathbf{B}$ is achieved by the NumPy function `np.kron(A,B)`.

2 Matrices and Their Properties

Transpose matrix I

Given a matrix $\mathbf{A} \in \mathbb{R}^{N \times M}$ the **transpose matrix** $\mathbf{A}^{\mathsf{T}} \in \mathbb{R}^{M \times N}$ is obtained by interchanging the rows and columns of $\mathbf{A}$

$$\mathbf{A}^{\mathsf{T}} = \begin{bmatrix} a_{11} & a_{21} & \cdots & a_{N1} \\ a_{12} & a_{11} & \cdots & a_{N2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1M} & a_{2M} & \cdots & a_{MN} \end{bmatrix}.$$

or

$$\mathbf{A}^{\mathsf{T}} = [a_{ji}], \qquad i = 1,\ 2,\ \ldots,\ N; \quad j = 1,\ 2,\ \ldots,\ M;$$

Hence, $\left(\mathbf{A}^{\mathsf{T}}\right)^{\mathsf{T}} = \mathbf{A}$.

Transposing a 2D signal is equivalent to swapping its dimensions.

$$\mathbf{X}^{\mathsf{T}} =$$

Figure 7: Matrix transpose of a 2D signal.

Hermitian matrix

Let us consider a matrix in the complex domain $\mathbf{A} \in \mathbb{K}^{N \times M}$, then it is possible to define the **Hermitian matrix** as a transpose and complex conjugate matrix

$$\mathbf{A}^{\mathsf{H}} = \left[a_{ji}^{*}\right], \qquad i = 1,\ 2,\ \ldots,\ N;\ \ j = 1,\ 2,\ \ldots,\ M.$$

In the real domain, $\mathbf{A}^{\mathsf{H}} = \mathbf{A}^{\mathsf{T}}$.

Remark

In Python, `A.conj().transpose()` or `A.conj().T` denotes a Hermitian transpose and `A.transpose()` or `A.T` a transpose.
In the real domain: `A.conj().T = A.T`.

Row of a matrix: Given a matrix $\mathbf{A} \in (\mathbb{R}, \mathbb{C})^{N \times M}$, its i-th *row vector*, with $i = 1, \dots, N$, is denoted as

$$\mathbf{a}_i \in (\mathbb{R}, \mathbb{C})^{1 \times M} = \begin{bmatrix} a_{i1} & a_{i2} & \cdots & a_{iM} \end{bmatrix}$$

Column of a matrix: Given a matrix $\mathbf{A} \in (\mathbb{R}, \mathbb{C})^{N \times M}$, its j-th *column vector*, with $j = 1, \dots, M$, is denoted as

$$\mathbf{a}_j \in (\mathbb{R}, \mathbb{C})^{N \times 1} = \begin{bmatrix} a_{1j} \\ a_{2j} \\ \vdots \\ a_{Nj} \end{bmatrix}.$$

Matrices as row or column vectors II

Remark

In Python you can extract entire columns or rows of a matrix with the following instructions:

- `A[(i-1)]` or `A[(i-1),:]`, extracts the entire i-th row in a row vector of dimension M;
- `A[(:,(j-1)]`, extracts the entire *j*-th column in column vector of size N;
- `A[0:K]` or `A[:K]` or `A[0:K,:]`, extracts the first K rows from A.
- `A[-K:]`, extracts the last K rows from A.

A matrix $\mathbf{A} \in (\mathbb{R},\mathbb{C})^{N\times M}$ can be represented by its N row vectors or by its M column vectors

$$\mathbf{A} = \begin{bmatrix} \mathbf{a}_1^{\mathsf{H}} \\ \mathbf{a}_2^{\mathsf{H}} \\ \vdots \\ \mathbf{a}_N^{\mathsf{H}} \end{bmatrix} = \begin{bmatrix} \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_N \end{bmatrix}^{\mathsf{H}}, \qquad \mathbf{A} = \begin{bmatrix} \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_M \end{bmatrix} = \begin{bmatrix} \mathbf{a}_1^{\mathsf{H}} \\ \mathbf{a}_2^{\mathsf{H}} \\ \vdots \\ \mathbf{a}_M^{\mathsf{H}} \end{bmatrix}^{\mathsf{H}}.$$

Then, given a matrix $\mathbf{A} \in (\mathbb{R},\mathbb{C})^{N\times M}$, we can associate a vector $\mathrm{vec}\,(\mathbf{A}) \in (\mathbb{R},\mathbb{C})^{NM\times 1}$ containing, all the stacked column vectors of $\mathbf{A}$

$$\begin{aligned} \mathrm{vec}\,(\mathbf{A}) &= \begin{bmatrix} \mathbf{a}_1^{\mathsf{H}} & \mathbf{a}_2^{\mathsf{H}} & \cdots & \mathbf{a}_M^{\mathsf{H}} \end{bmatrix}^{\mathsf{H}}_{M(N)\times 1} \\ &= [a_{11}, ..., a_{N1},\; a_{12}, ..., a_{N2}, \cdots\cdots, a_{1M}, ..., a_{NM}]^{\mathsf{H}}_{NM\times 1}. \end{aligned}$$

In Python, reshaping is performed by the NumPy function `np.reshape(A,(N,M))`.

Diagonal matrix

A given matrix $\mathbf{A} \in (\mathbb{R}, \mathbb{C})^{N \times N}$ is called **diagonal** if $a_{ji} = 0$ for $i \neq j$.

In Python, the `diag` operator yields both a diagonal matrix from a vector, e.g., `A = diag(a);`

$$\mathbf{A} = \operatorname{diag}(\mathbf{a}) = \operatorname{diag}\begin{bmatrix} a_1 \\ \vdots \\ a_N \end{bmatrix} = \begin{bmatrix} a_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & a_N \end{bmatrix}$$

and a vector from a matrix, e.g., `a = diag(A);`

$$\mathbf{a} = \operatorname{diag}^{-1}(\mathbf{A}) = \operatorname{diag}^{-1}\begin{bmatrix} a_{11} & \cdots & a_{1N} \\ \vdots & \ddots & \vdots \\ a_{N1} & \cdots & a_{NN} \end{bmatrix} = \begin{bmatrix} a_{11} \\ \vdots \\ a_{NN} \end{bmatrix}$$

Trace and symmetry

The **trace** of a matrix $\mathbf{A} \in (\mathbb{R}, \mathbb{C})^{N \times N}$ is given by the sum of its diagonal elements

$$\text{tr}\left(\mathbf{A}\right) = \text{tr}\begin{bmatrix} a_{11} & \cdots & a_{1N} \\ \vdots & \ddots & \vdots \\ a_{N1} & \cdots & a_{NN} \end{bmatrix} = \sum_{i=1}^{N} a_{ii}$$

In Python, this can be expressed as `A.trace`.

A matrix $\mathbf{A} \in (\mathbb{R}, \mathbb{C})^{N \times N}$ is **symmetric** if $a_{ji} = a_{ij}$ or $a_{ji} = a_{ij}^*$ in the complex-valued domain, whereby $\mathbf{A}^{\mathsf{T}} = \mathbf{A}$ for real domain and $\mathbf{A}^{\mathsf{H}} = \mathbf{A}$ for the complex-valued domain.

Toeplitz matrix

$\mathbf{A} \in (\mathbb{R}, \mathbb{C})^{N \times N}$ is a **Toeplitz matrix** if $[a_{i,j}] = [a_{i+1,j+1}] = [a_{i-j}]$, i.e., each descending diagonal of $\mathbf{A}$, from left to right, is constant.

$$\mathbf{A}_T = \begin{bmatrix} a_i & a_{i-1} & a_{i-2} & a_{i-3} & \cdots \\ a_{i+1} & a_i & a_{i-1} & a_{i-2} & \ddots \\ a_{i+2} & a_{i+1} & a_i & a_{i-1} & \ddots \\ a_{i+3} & a_{i+2} & a_{i+1} & a_i & \ddots \\ \vdots & \ddots & \ddots & \ddots & \ddots \end{bmatrix}$$

In Python, from scipy.linalg, `A = toeplitz(c,r)` returns a *nonsymmetric* Toeplitz matrix with `c` as its first column and `r` as its first row.

A *symmetric* Toeplitz matrix is simply created by `A = toeplitz(c)`.

Positive semidefinite matrix

A matrix $\mathbf{A} \in \mathbb{K}^{N \times N}$ is **positive semidefinite**, or *nonnegative*, if

$$\forall \mathbf{w} \in \mathbb{K}^{N} \Rightarrow \Re\left(\mathbf{w}^{\mathsf{H}} \mathbf{A} \mathbf{w}\right) \geq 0.$$

A positive semidefinite matrix can be also denoted as $\mathbf{A} \succeq 0$.
The symbol $\succeq$ represents inequality between matrices, i.e., given two matrices $(\mathbf{A}, \mathbf{B}) \in (\mathbb{R}, \mathbb{C})^{N \times N}$, $\mathbf{A} \succeq \mathbf{B}$ means that

$$\mathbf{z}^{\mathsf{T}} \mathbf{A} \mathbf{z} \geq \mathbf{z}^{\mathsf{T}} \mathbf{B} \mathbf{z}, \quad \forall \mathbf{z} \in (\mathbb{R}, \mathbb{C})^{N \times 1}.$$

If $\mathbf{w}^{\mathsf{H}} \mathbf{A} \mathbf{w} > 0$, then the matrix is referred to as positive definite $\mathbf{A} \succ 0$.

Inverse matrix

A square matrix $\mathbf{A} \in (\mathbb{R}, \mathbb{C})^{N\times N}$ is **invertible**, or *nonsingular*, if there exists a matrix $\mathbf{B} \in (\mathbb{R}, \mathbb{C})^{N\times N}$ such that

$$\mathbf{B}\mathbf{A} = \mathbf{I},$$

where $\mathbf{I}_{N\times N}$ is an *identity matrix* defined as

$$\mathbf{I} = \text{diag}\,(1,\ 1, \ldots\ , 1)\,.$$

In such a case, the matrix $\mathbf{B}$ is uniquely determined from $\mathbf{A}$ and defined as the **inverse** of $\mathbf{A}$, or also denoted as $\mathbf{A}^{-1}$. This implies:

$$\mathbf{A}^{-1}\mathbf{A} = \mathbf{I}.$$

In Python, $\mathbf{A}^{-1}$ is performed by `linalg.inv(A)`.

Some properties of inverse matrices

It is worth noting that if $\mathbf{A}$ is nonsingular, the system equations

$$\mathbf{A}\mathbf{x} = \mathbf{b}$$

has a unique solution given by $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$. In Python: `linalg.solve(A,B)` if `A` is square; `linalg.lstsq(A,B)` otherwise.

Property

$$\begin{aligned}
(\mathbf{A}\mathbf{B}\mathbf{C}\cdots)^{-1} &= \cdots\mathbf{C}^{-1}\mathbf{B}^{-1}\mathbf{A}^{-1} \\
\left(\mathbf{A}^{H}\right)^{-1} &= \left(\mathbf{A}^{-1}\right)^{H} \\
(\mathbf{A}+\mathbf{B})^{H} &= \mathbf{A}^{H}+\mathbf{B}^{H} \\
(\mathbf{A}\mathbf{B})^{H} &= \mathbf{B}^{H}\mathbf{A}^{H} \\
(\mathbf{A}\mathbf{B}\mathbf{C}\cdots)^{H} &= \cdots\mathbf{C}^{H}\mathbf{B}^{H}\mathbf{A}^{H}.
\end{aligned}$$

Pseudoinverse matrix

The *generalized inverse* matrix, or **Moore-Penrose pseudoinverse**, represents a general way to determine of the solution of a linear real or complex system equations $\mathbf{A}\mathbf{x} = \mathbf{b}$, when $\mathbf{A}$ is a rectangular matrix, i.e., $\mathbf{A} \in (\mathbb{R}, \mathbb{C})^{N\times M}$, $\mathbf{x} \in (\mathbb{R}, \mathbb{C})^{M\times 1}$, $\mathbf{b} \in (\mathbb{R}, \mathbb{C})^{N\times 1}$.

The pseudoinverse of a generic rectangular matrix $\mathbf{A} \in (\mathbb{R}, \mathbb{C})^{N\times M}$ is denoted as $\mathbf{A}^{\#}$ and it is characterized by the following properties.

Property

$$
\begin{aligned}
\mathbf{A}\mathbf{A}^{\#}\mathbf{A} &= \mathbf{A} \\
\mathbf{A}^{\#}\mathbf{A}\mathbf{A}^{\#} &= \mathbf{A}^{\#} \\
\mathbf{A}\mathbf{A}^{\#} &= \left(\mathbf{A}\mathbf{A}^{\#}\right)^{H} \\
\mathbf{A}^{\#}\mathbf{A} &= (\mathbf{A}^{\#}\mathbf{A})^{H}.
\end{aligned}
$$

Solution by a pseudoinverse matrix

The solution of a general linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$ depends on the size of the matrix $\mathbf{A}$:

$$\mathbf{A}^{\#} = \begin{cases} \mathbf{A}^{-1} & N = M & \text{square matrix} \\ \mathbf{A}^{\mathsf{H}} \left(\mathbf{A}\mathbf{A}^{\mathsf{H}}\right)^{-1} & N < M & \text{“fat” matrix} \\ \left(\mathbf{A}^{\mathsf{H}}\mathbf{A}\right)^{-1} \mathbf{A}^{\mathsf{H}} & N > M & \text{“tall” matrix.} \end{cases}$$

Anyway, the solution of $\mathbf{A}\mathbf{x} = \mathbf{b}$ may always be expressed as

$$\mathbf{x} = \mathbf{A}^{\#}\mathbf{b}.$$

In Python, $\mathbf{A}^{\#}$ is performed by `linalg.pinv(A)`, while the solution can be obtained by `x = linalg.pinv(A)@b = linalg.solve(A,B)` for $\mathbf{A}$ square, or `x = linalg.pinv(A)@b = linalg.lstsq(A,B)` otherwise.

Different methods for calculating the pseudoinverse refer to possible decompositions of the matrix $\mathbf{A}$.

Matrix inversion lemma

The **matrix inversion lemma** (MIL) is a very useful property in the development of online machine learning algorithms.

The MIL states that if $\mathbf{A}^{-1}$ and $\mathbf{C}^{-1}$ exist, the following equation is true (as well as several equivalent variants):

$$[\mathbf{A}+\mathbf{BCD}]^{-1}=\mathbf{A}^{-1}-\mathbf{A}^{-1}\mathbf{B}\left[\mathbf{C}^{-1}+\mathbf{DA}^{-1}\mathbf{B}\right]^{-1}\mathbf{DA}^{-1}$$

where $\mathbf{A}\in\mathbb{K}^{M\times M}$, $\mathbf{B}\in\mathbb{K}^{M\times N}$, $\mathbf{C}\in\mathbb{K}^{N\times N}$ and $\mathbf{D}\in\mathbb{K}^{N\times M}$.

The **rank** r of a matrix $\mathbf{A} \in \mathbb{R}^{M \times N}$ represents the number of independent columns of $\mathbf{A}$.

It is useful in linear system equations to understand if a system admits a solution (i.e., *the rank of the coefficient matrix must be the same of the complete matrix including the constant terms*).

An algebraic tool to easily determine the rank of any matrix is the determinant.

The **determinant** of a matrix $\mathbf{A}$ is denoted as $\det(\mathbf{A})$ or $\Delta_{\mathbf{A}}$, and it can be computed by using several algorithms according to the size of the matrix.

Rank and determinant of a matrix II

The determinant of a square matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ is defined in terms of the determinant of order $N-1$ with the following recursive expression

$$\det(\mathbf{A}) = \sum_{j=1}^{N} a_{ij} \left[(-1)^{j+i} \det(\mathbf{A}_{ij}) \right]$$

where $\mathbf{A}_{ij} \in \mathbb{R}^{(N-1)\times(N-1)}$ is a matrix obtained by eliminating the i-th row and the j-th column of $\mathbf{A}$.

A matrix $\mathbf{A}$ with $\det(\mathbf{A}) \neq 0$ is called *nonsingular* and it is always invertible.

Note that the determinant of a diagonal or triangular matrix is the product of the values on the diagonal.

Low-rank approximation

In several machine learning problems, low-rank approximation methods are used to describe a large matrix $\mathbf{Y} \in \mathbb{R}^{M \times N}$ with a smaller one $\mathbf{X} \in \mathbb{R}^{R \times N}$, with $R < M$:

$$\mathbf{Y} \approx \mathbf{A}\mathbf{X}$$

being $\mathbf{A} \in \mathbb{R}^{M \times R}$.

Note

In problems of data compression, very often the minimization of a cost function that measures the fit between a given data matrix and an approximating matrix is formulated under a low-rank approximation constraint, i.e., the approximating matrix is constrained to have a reduced rank. This kind of constraint is equivalent to reducing the complexity of a model that fits the data.

3 Geometrical Properties of Vectors

Inner and Outer Products

Projection Operators

Inner and outer products

Given two vectors $\mathbf{u} \in (\mathbb{R}, \mathbb{C})^{N\times 1}$ and $\mathbf{v} \in (\mathbb{R}, \mathbb{C})^{N\times 1}$, the *inner product*, or *scalar product* or **dot product**, denoted as $\langle \mathbf{u}, \mathbf{v} \rangle$, is defined as

$$\langle \mathbf{u}, \mathbf{v} \rangle = \mathbf{u}^{\mathsf{H}}\mathbf{v} = \left(\mathbf{u}^{\mathsf{H}}\mathbf{v}\right)^* = \sum_{i=1}^{N} u_i v_i^* .$$

The **outer product** between two vectors $\mathbf{u} \in (\mathbb{R}, \mathbb{C})^{M\times 1}$ and $\mathbf{v} \in (\mathbb{R}, \mathbb{C})^{N\times 1}$, denoted as $\rangle \, \mathbf{u}, \mathbf{v} \, \langle \, \in (\mathbb{R}, \mathbb{C})^{M\times N}$, is a matrix defined by the product

$$\rangle \, \mathbf{u}, \mathbf{v} \, \langle = \mathbf{u}\mathbf{v}^{\mathsf{H}} = \begin{bmatrix} u_1 v_1^* & \cdots & u_1 v_N^* \\ \vdots & \ddots & \vdots \\ u_M v_1^* & \cdots & u_M v_N^* \end{bmatrix}_{M\times N}$$

The inner product represents a basic operation in many NNs models.

A neuron of a NN receives inputs x_i, $i = 1 \ldots, N$, from the N nodes of the previous layer; each input x_i is weighted with its own coefficient w_i, thus:

$$s = w_1 x_1 + w_2 x_2 + \ldots + w_N x_N = \sum_{i=1}^{N} w_i x_i = \mathbf{x}^\mathsf{T}\mathbf{w} = \mathbf{w}^\mathsf{T}\mathbf{x}$$

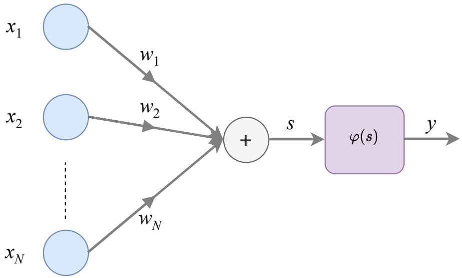


Figure 8: Graphical representation of a neuron receiving N inputs frm the previous layer.

Geometrical interpretation I

The inner product of a vector for itself $\mathbf{u}^{\mathsf{H}}\mathbf{u}$, can be written as

$$\langle \mathbf{u}, \mathbf{u} \rangle = \mathbf{u}^{\mathsf{H}}\mathbf{u} \triangleq \|\mathbf{u}\|_2^2$$

which defines the square of its length in a Euclidean space.

In Euclidean geometry, the inner product of vectors expressed in an orthonormal basis is related to their *length* and *angle*.

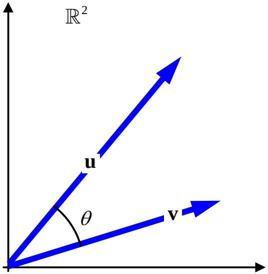
Considering two vectors $\mathbf{u}$ and $\mathbf{v}$, their lengths can be expressed as

$$\|\mathbf{u}\| \triangleq \sqrt{\|\mathbf{u}\|_2^2}, \qquad \|\mathbf{v}\| \triangleq \sqrt{\|\mathbf{v}\|_2^2}$$

Therefore, the inner product can be also expressed as

$$\mathbf{u}^{\mathsf{H}}\mathbf{v} = \|\mathbf{u}\| \cdot \|\mathbf{v}\| \cos\theta$$

where θ is the angle between $\mathbf{u}$ and $\mathbf{v}$.



Inner product

provides X with a structure can be viewed as a '*similarity*'

- $\langle \mathbf{u}, \mathbf{v} \rangle > 0$, $\mathbf{u}$ and $\mathbf{v}$ point to the 'same direction'
- $\langle \mathbf{u}, \mathbf{v} \rangle < 0$, $\mathbf{u}$ and $\mathbf{v}$ point to the 'opposite direction'
- $\langle \mathbf{u}, \mathbf{v} \rangle = 0$, $\mathbf{u}$ and $\mathbf{v}$ are orthogonal, $\mathbf{u} \perp \mathbf{v}$.

Figure 9: Graphical representation of the inner product.

Orthogonality

Whenever dealing with signals, the conditions of **orthogonality** (but also orthonormality and bi-orthogonality) represent a fundamental tool to understand some characterizing properties of signals, e.g., correlations between signals, convergence properties, and much more.

Two vectors (e.g., two signals) $\mathbf{x}, \mathbf{w} \in (\mathbb{R}, \mathbb{C})^N$ are orthogonal if their inner product is zero $\langle \mathbf{x}, \mathbf{w} \rangle = 0$.
This is sometimes also referred to as $\mathbf{x} \perp \mathbf{w}$.

This concept can be extended to a set of vectors $\{\mathbf{a}_i\} \triangleq \{\mathbf{a}_i \in (\mathbb{R}, \mathbb{C})^N, \ \ \forall i, \ i = 1, ..., N\}$, which is called *orthogonal* if

$$\mathbf{a}_i^{\mathsf{H}} \mathbf{a}_j = 0, \qquad i \neq j.$$

Orthonormality

A set of vectors $\{\mathbf{a}_i\}$, is *orthonormal* if

$$\mathbf{a}_i^{\mathsf{H}}\mathbf{a}_j = \delta_{ij} = \delta\left[i - j\right]$$

where δ_{ij} is the Kronecker symbol defined as: $\delta_{ij} = 1$ for $i = j$; $\delta_{ij} = 0$ for $i \neq j$.

A matrix $\mathbf{A} \in \mathbb{K}^{N\times N}$ is orthonormal if its columns are an orthonormal set of vectors.

Formally, a matrix $\mathbf{A} \in \mathbb{K}^{N\times N}$ is an **orthogonal matrix** if

$$\mathbf{A}\mathbf{A}^{\mathsf{T}} = \mathbf{I},$$

where $\mathbf{A}^{\mathsf{T}}$ is the transpose of $\mathbf{A}$ and $\mathbf{I}$ is the *identity* matrix.

Properties of orthonormal matrices

An orthogonal matrix is always *invertible*.

Moreover, $\mathbf{A}^{-1} = \mathbf{A}^{\mathsf{T}}$, i.e., in component form, $a_{ij}^{-1} = a_{ji}$.

These properties make orthogonal matrices particularly easy to compute, since the transpose operation is much simpler than computing a matrix inversion.

Orthonormality has no effect on inner product, i.e.

$$\langle \mathbf{Ax}, \mathbf{Ay} \rangle = (\mathbf{Ax})^{\mathsf{H}} \mathbf{Ay} = \mathbf{x}^{\mathsf{H}} \mathbf{A}^{\mathsf{H}} \mathbf{Ay} = \langle \mathbf{x}, \mathbf{y} \rangle$$

Furthermore, if multiplied to a vector, it does not change its length

$$\|\mathbf{Ax}\|_2^2 = (\mathbf{Ax})^{\mathsf{H}} \mathbf{Ax} = \mathbf{x}^{\mathsf{H}} \mathbf{A}^{\mathsf{H}} \mathbf{Ax} = \|\mathbf{x}\|_2^2 .$$

Projection matrix I

A square matrix $\mathbf{P} \in (\mathbb{R}, \mathbb{C})^{N \times N}$ is a **projection operator** iff $\mathbf{P}^2 = \mathbf{P}$, i.e., it is *idempotent*.

If $\mathbf{P}$ is symmetric, then the projection is orthogonal.

If $\mathbf{P}$ is a projection matrix, $(\mathbf{I} - \mathbf{P})$ is also a projection matrix.

Examples of orthogonal projection matrices are matrices associated with the pseudoinverse $\mathbf{A}^{\#}$ in the over- and under-determined cases.

Projection matrix II

In the overdetermined case, $N > M$ and $\mathbf{A}^{\#} = \left(\mathbf{A}^{\mathsf{H}}\mathbf{A}\right)^{-1}\mathbf{A}^{\mathsf{H}}$, so the orthogonal projection operator can be defined as

$$\mathbf{P} = \mathbf{A}\left(\mathbf{A}^{\mathsf{H}}\mathbf{A}\right)^{-1}\mathbf{A}^{\mathsf{H}}$$

On the other hand, the *orthogonal complement projection operator* is defined as $\mathbf{P}^{\perp} = \mathbf{I} - \mathbf{A}\left(\mathbf{A}^{\mathsf{H}}\mathbf{A}\right)^{-1}\mathbf{A}^{\mathsf{H}}$, so that $\mathbf{P} + \mathbf{P}^{\perp} = \mathbf{I}$.

Remark

The operator $\mathbf{P}$ projects a vector on the subspace $\Psi = \mathcal{R}\left(\mathbf{A}\right)$, while $\mathbf{P}^{\perp}$ on its orthogonal complement $\Psi^{\perp} = \mathcal{R}^{\perp}\left(\mathbf{A}\right)$ or $\sum = \mathcal{N}\left(\mathbf{A}^{\mathsf{H}}\right)$.

Indeed, given $\mathbf{x} \in (\mathbb{R}, \mathbb{C})^{M\times 1}$ and $\mathbf{y} \in (\mathbb{R}, \mathbb{C})^{N\times 1}$, such that $\mathbf{Ax} = \mathbf{y}$, we have that $\mathbf{Py} = \mathbf{u}$ and $\mathbf{P}^{\perp}\mathbf{y} = \mathbf{v}$ such that $\mathbf{u} \in \mathcal{R}\left(\mathbf{A}\right)$ and $\mathbf{v} \in \mathcal{N}\left(\mathbf{A}^{\mathsf{H}}\right)$.
In the underdetermined case, where $N < M$ and $\mathbf{A}^{\#} = \mathbf{A}^{\mathsf{H}}\left(\mathbf{A}\mathbf{A}^{\mathsf{H}}\right)^{-1}$, we have

$$\mathbf{P} = \mathbf{A}^{\mathsf{H}}\left(\mathbf{A}^{\mathsf{H}}\mathbf{A}\right)^{-1}\mathbf{A} \qquad \text{with} \quad \mathbf{P}^{\perp} = \mathbf{I} - \mathbf{A}^{\mathsf{H}}\left(\mathbf{A}\mathbf{A}^{\mathsf{H}}\right)^{-1}\mathbf{A}.$$

$$\mathbf{P} = \mathbf{A}(\mathbf{A}^H\mathbf{A})^{-1}\mathbf{A}^H$$

$$\mathbf{P}^{\perp} = \mathbf{I} - \mathbf{A}(\mathbf{A}^H\mathbf{A})^{-1}\mathbf{A}^H$$

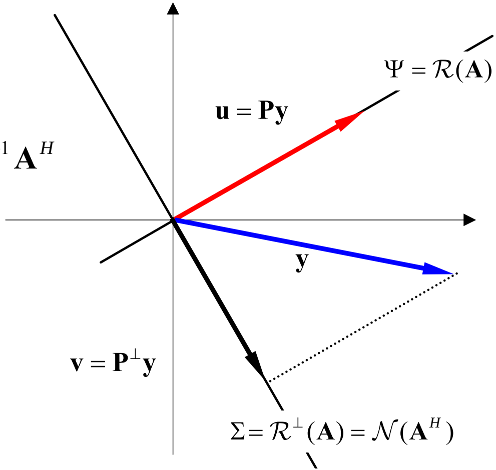


Figure 10: Representation of the orthogonal projection operator.

We consider an audio signal, in this case a gong, and a musical note A. We want to know "how much" of the note A is contained in the gong.

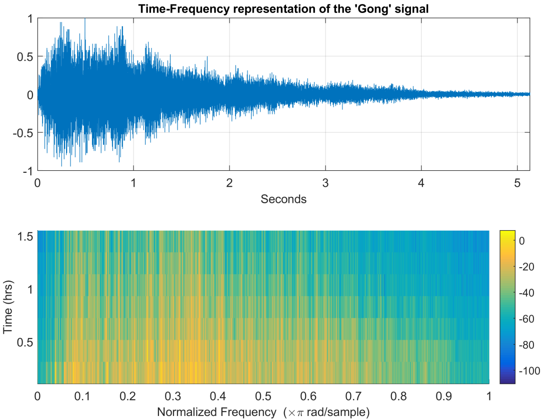


Figure 11: Representation of the orthogonal projection operator.

Example: projection of audio signals II

To this end we project the original signal into the musical note A.

We represent the gong signal $\mathbf{x}$ and the note signal $\mathbf{a}$ in the time-frequency domain by applying a short-time Fourier transform (SFTF):

$$\mathbf{X} = \text{STFT}\left\{\mathbf{x}\right\}, \qquad \mathbf{A} = \text{STFT}\left\{\mathbf{a}\right\}.$$

Then, we compute the projection matrix and the projected signal in its time-frequency domain:

$$\mathbf{P} = \mathbf{A}\left(\mathbf{A}^{\mathsf{H}}\mathbf{A}\right)^{-1}\mathbf{A}^{\mathsf{H}}$$
$$\mathbf{U} = \mathbf{P}\mathbf{X}$$

Example: projection of audio signals III

By anti-transforming $\mathbf{U}$ in the time domain, we achieve the resulting signal containing the information of the musical note A in the gong signal, i.e., $\mathbf{u} = \text{STFT}^{-1}\{\mathbf{U}\}$.

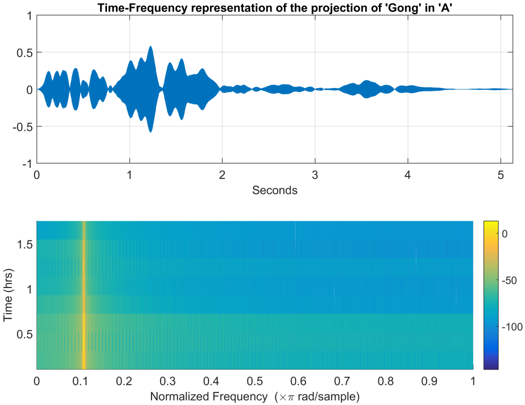


Figure 12: Representation of the orthogonal projection of the gong signal in the musical note A.

4 Metrics and Norms

Distance Metrics

Norm of Vectors

Notable Inequalities

Norm of Matrices

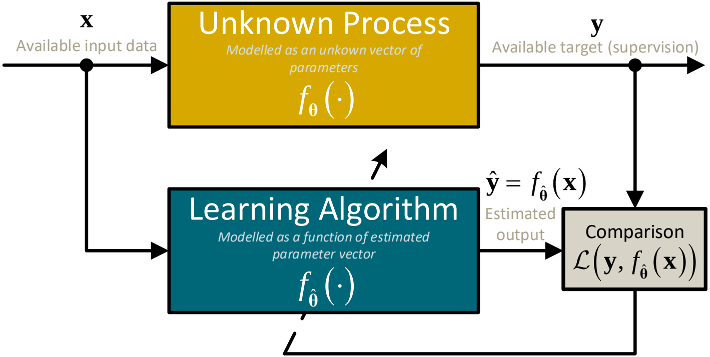


Figure 13: The unknown system is modelled as a function of estimated parameters. Learning is performed after comparing the target signal $\mathbf{y}$ (for *supervised learning*) and the estimated output $\hat{\mathbf{y}} = f_{\hat{\boldsymbol{\theta}}}(\cdot)$. Now we focus on how to measure the distance between the target signal (or also denoted as *ground truth*) and the estimated output.

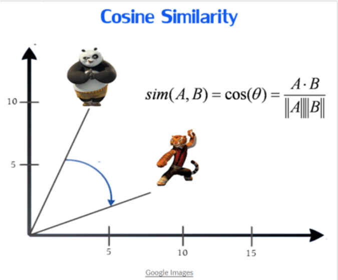
Distance metrics in learning algorithms

Why Distance Metrics are important in learning algorithms?

- In machine learning, many supervised and unsupervised algorithms use Distance Metrics to analyze input data patterns.
- Distance Metrics are used to recognize similarities among patterns.
- Distance Metrics are used to define the “rewarding function” of learning algorithm (usually called *loss functions*, e.g., the mean squared error.

The choice of a “good” metric is fundamental to achieve optimal performance in classification, regression and clustering algorithms.

Word2Vec (W2V) is a neural network that preprocesses text.



Word	Cosine distance
spain	0.678515
belgium	0.665923
netherlands	0.652428
italy	0.633130
switzerland	0.622323
luxembourg	0.610033
portugal	0.577154
russia	0.571507
germany	0.563291
catalonia	0.534176

Figure 14: Example of distances from the word “france”.

Vector spaces

A **Vector Space** $(X, \mathbb{K})$ is a collection of objects called vectors $\mathbf{x}, \mathbf{y}, \mathbf{z}, \ldots \in X$, (e.g., arrays, continuous or discrete functions, polynomials,...) that can be added together and multiplied by scalars $c \in \mathbb{K}$.

A **Metric Vector Space**, denoted as $(X, \mathbb{K}, d)$, or as (X, d), is a vector space equipped with a strictly positive function $d : X \times X \to R$, denoted **distance**, that assigns a *length* (i.e., a measure) to each vector in a vector space.

A **Normed Vector Space** $(X, \|\cdot\|)$ is a vector space over the real or complex numbers on which the distance is a *norm*.

Examples of vector spaces

- **Banach spaces** $(B, \|\cdot\|)$ is a complete *vector space* where the metric is the norm $d = \|\mathbf{x} - \mathbf{y}|$.

- **Hilbert spaces** $(H, < \cdot, \cdot >)$ is a complete *inner product space* where the norm is induced by the inner product $d =< \mathbf{x}, \mathbf{y} >$.

- **Reproducing kernel Hilbert space** (RKHS) is a Hilbert space where functionals with particular characteristics of *smoothness* are defined, which make them suitable for functions approximation/interpolation problems.

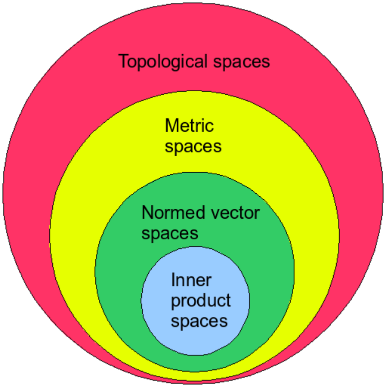


Figure 15: Hierarchy of mathematical spaces. Normed vector spaces are a superset of inner product spaces and a subset of metric spaces, which in turn is a subset of topological spaces. Hilbert spaces are subsets of iner product spaces as well as the Euclidean vetor space. Source: Wikipedia.

Norm of vectors

Together with the scalar product, the **norm** operator is one the most natural ways to define or measure the distance in a *metric space*.

Given a vector $\mathbf{x} \in (\mathbb{R}, \mathbb{C})^N$, its *norm* refers to its length with respect to a vector space.

In the case of a normed-space of order p, said ℓ_p-*space*, the norm is denoted as $\|\mathbf{x}\|^{\ell_p}$ or $\|\mathbf{x}\|_p$, and it is defined as:

Definition (ℓ_p Vector Norm)

A generic norm of a vector is defined as

$$\|\mathbf{x}\|_p \triangleq \left[\sum_{i=1}^{N} |x_i|^p\right]^{1/p}, \quad p \geq 1.$$

ℓ_0 vector "norm"

The previous definition of the ℓ_p vector norm is valid even when $0 < p < 1$; however, the results have a different meaning.

Particular attention is paid to ℓ_0, ℓ_1, ℓ_2 and ℓ_∞ norms.

Definition (ℓ_0 Vector Norm)

The ℓ_p vector norm, for $p = 0$, can be expressed as

$$\|\mathbf{x}\|_0 \triangleq \lim_{p\to 0} \|\mathbf{x}\|_p = \sum_{i=1}^{N} |x_i|^0$$

that measures the *numerosity* of non-zero elements, i.e., the number of non-zero entries of the vector $\mathbf{x}$.

ℓ_1 and ℓ_∞ vector norms

Definition (ℓ_1 Vector Norm)

For $p = 1$, the vector norm becomes

$$\|\mathbf{x}\|_1 \triangleq \sum_{i=1}^{N} |x_i|,$$

which represents the sum of the absolute values of the elements of $\mathbf{x}$.

Definition (ℓ_∞ Vector Norm)

For $p \to \infty$, the vector norm becomes

$$\|\mathbf{x}\|_\infty \triangleq \max_{i=1,N} \{|x_i|\},$$

which is also known as *uniform norm*, *supremum norm* (or sup norm), *Chebyshev norm*, or *infinity norm*.

Definition (ℓ_2 Vector Norm)

The norm is defined for $p = 2$, also known as **Euclidean norm**, expresses the standard length of the vector

$$\|\mathbf{x}\|_2 \triangleq \sqrt{\sum_{i=1}^{N} |x|_i^2} = \sqrt{\mathbf{x}^{\mathsf{H}}\mathbf{x}}$$

$$\|\mathbf{x}\|_2^2 \triangleq \mathbf{x}^{\mathsf{H}}\mathbf{x}$$

Let $\mathbf{G}$ be a suited diagonal matrix, said weighing matrix, the expression:

$$\|\mathbf{x}\|_{\mathbf{G}}^2 \triangleq \left\|\mathbf{x}^{\mathsf{H}}\mathbf{G}\mathbf{x}\right\|$$

represents the so called weighted norm.

Distance measures

Property

The distance between two vectors $\mathbf{x}$ *and* $\mathbf{y}$ *is defined as*

$$\|\mathbf{x}-\mathbf{y}\|_p \triangleq \left[\sum_{i=1}^{N} |x_i - y_i|^p\right]^{1/p}, \quad p > 0$$

and represents a similarity measure in the Minkowsky metric.

Definition (Manhattan Distance)

The Manhattan (or *taxicab*) distance between two vectors $\mathbf{x}$ and $\mathbf{y}$ is:

$$\|\mathbf{x}-\mathbf{y}\|_1 = \sum_{i=1}^{N} |x_i - y_i|.$$

In a Cartesian system, it represents the sum of the lengths of the projections of the segments between the points onto the coordinate axes.

Hölder's inequality

The triangular and Cauchy-Schwarz inequalities can be generalized to the case of norm of order $p \in [1, \infty]$.

The **Hölder's inequality**, which is extension of the Cauchy-Schwarz's inequality, can be defined by the following theorem.

Theorem (Hölder's Inequality)

Given two vector $\mathbf{x}, \mathbf{y} \in (\mathbb{R}, \mathbb{C})^N$ with possible infinite length, let $p, q \in [1, \infty)$ with $1/p + 1/q = 1$, the following identity holds

$$\left| \sum_{i=1}^{\infty} x_i^\star y_i \right| \leq \left[\sum_{i=1}^{\infty} |x_i|^p \right]^{1/p} \cdot \left[\sum_{i=1}^{\infty} |y_i|^q \right]^{1/q}$$

$$\text{i.e.,} \qquad \left| \mathbf{x}^H \mathbf{y} \right| \leq \|\mathbf{x}\|_p \cdot \|\mathbf{y}\|_q$$

with the equality that applies only if $\mathbf{x}$ and $\mathbf{y}$ are linearly dependent (i.e., lie on the same line).

Minkowski's inequality

Using the Hölder's inequality, we can establish the triangle inequality for ℓ_p norms denoted as **Minkowski's inequality** and defined as follows.

Theorem (Minkowski's Inequality)

Given two vector $\mathbf{x}, \mathbf{y} \in (\mathbb{R}, \mathbb{C})^N$ *with possible infinite length, let* $p \in [1, \infty)$*, we have that*

$$\|\mathbf{x} + \mathbf{y}\|_p \leq \|\mathbf{x}\|_p + \|\mathbf{y}\|_p$$

The norm of a matrix can be defined similarly to the vector norms. Let us consider a matrix $\mathbf{A} \in (\mathbb{R}, \mathbb{C})^{N \times M}$, the matrix norm can be defined as follows.

Definition (ℓ_1 Matrix Norm)

$$\|\mathbf{A}\|_1 \triangleq \max_{j \in [1, M]} \sum_{i=1}^{N} |a_{ij}|$$

i.e., $\|\mathbf{A}\|_1$ represents the maximum absolute column sum of $\mathbf{A}$.

Definition (ℓ_2 Matrix Norm)

The **Euclidean norm** is defined for the space $p = 2$ and it can be expresses as

$$\|\mathbf{A}\|_2 \triangleq \sqrt{\lambda_{\max}} \quad \Rightarrow \quad \max_{\lambda_i} \left[\text{eig}\left(\mathbf{A}^{\mathsf{H}}\mathbf{A}\right)\right] \subseteq \max_{\lambda_i} \left[\text{eig}\left(\mathbf{A}\mathbf{A}^{\mathsf{H}}\right)\right]$$

Definition (Frobenius Matrix Norm)

$$\|\mathbf{A}\|_{\mathsf{F}} \triangleq \sqrt{\sum_{i=1}^{N}\sum_{j=1}^{M} |a_{ij}|^2} = \sqrt{\text{tr}\left(\mathbf{A}\mathbf{A}^{\mathsf{H}}\right)} = \sqrt{\sum_{i=1}^{\min(N,M)} \sigma_i^2}$$

where $\text{tr}\left(\mathbf{A}\mathbf{A}^{\mathsf{H}}\right)$ is the trace of $\mathbf{A}\mathbf{A}^{\mathsf{H}}$, and σ_i are the singular values of the matrix $\mathbf{A}$.

Note that the Frobenius norm is nothing but the Euclidean norm applied to $\text{vec}\,(\mathbf{A})$.

Definition (ℓ_∞ Matrix Norm)

Given a $\mathbf{A}_{N\times M}$ matrix the ℓ_∞ matrix norm is defined as

$$\|\mathbf{A}\|_\infty \triangleq \max_{i\in[1,N]} \sum_{j=1}^{M} |a_{ij}|$$

i.e. $\|\mathbf{A}\|_\infty$ represents the maximum absolute row sum of $\mathbf{A}$.

Next lecture

- We introduce elements of unconstrained convex optimization.
- We focus on one of the most popular cost functions in learning algorithms that is the least-square cost function.
- We introduce the gradient operator that is strictly related to the concept of *differentiability* in neural networks.

References

[1] G. H. Golub and C. F. Van Loan, *Matrix Computation*.
Baltimore and London: John Hopkins University Press, 1989.

[2] B. Raj, "Machine Learning for Signal Processing course," 2013.
Carnegie Mellon University.

[3] P. Smaragdis, "Machine Learning for Signal Processing course," 2018.
University of Illinois at Urbana-Champaign.

[4] G. Strang, *Introduction to Linear Algebra*.
Wellesley-Cambridge Press, 5th ed., 2016.

[5] A. Uncini, *Machine Learning Mathematical Elements: Functional Analysis, Nonlinear Programming, Stochastic Processes and Estimation Theory*.
2017.

[6] K. B. Petersen and M. S. Pedersen, *The Matrix Cookbook*.
Technical University of Denmark, 2012.

Elements of Linear Algebra

Neural Networks 2023/2024

Danilo Comminiello

https://sites.google.com/uniroma1.it/neuralnetworks2023

{danilo.comminiello, simone.scardapane}@uniroma1.it