1. Preliminaries

This Chapter provides a selection of preliminary notions for the contents of this Thesis. We provide CAGD and DL definitions, notations, and operations that will be essential throughout the following Chapters. After introducing univariate B-spline constructions and their tensor-product multivariate extensions, we focus on the hierarchical spline model by considering THB-splines. Subsequently, we illustrate NN architectures and mainly focus on CNNs and GCNs. The present Chapter is mostly based on [11, 55, 64, 19].

1.1. Adaptive spline constructions

CAGD is concerned with the mathematical and computational methods for geometric modelling. This discipline is primarily devoted to the construction and analysis of free-form curves, surfaces, and volumes, such as splines, meshes, and subdivision surfaces, as well as suitable modelling and/or approximation schemes for their generation, analysis, and manipulation [76, 44]. The range of CAGD applications is very wide, e. g., CAD, Computer Aided Engineering (CAE)/CAM, computer graphics, path planning and motion control, robotics, and scientific visualization, among others.

The CAGD methods considered in this Thesis concern splines and related approximation and modelling schemes. The computational CAGD tools employed are B-splines, piecewise polynomial functions with a certain regularity, and their multivariate adaptive extensions. In particular, in this Section we present the main concepts of polynomial B-splines and truncated hierarchical B-splines.

B-splines

The choice of the approximation space plays a fundamental role in the final accuracy of the geometric model. This is the case of polynomial interpolation models, which might be affected by oscillations for too high polynomial degree [139]. A solution to this issue is provided by considering spline models, which allow to keep the polynomial degree low, while improving the model accuracy by increasing the number of polynomial pieces for its definition.

Sofia Imperatore, Eindhoven University of Technology, the Netherlands, s.imperatore@tue.nl, 0009-0003-9116-9978

Referee List (DOI 10.36253/fup_referee_list)
FUP Best Practice in Scholarly Publishing (DOI 10.36253/fup_best_practice)

Let $\Omega = [a, b] \subset \mathbb{R}$ be a real interval and let $\boldsymbol{\tau} \coloneqq \{\tau_0, \ldots, \tau_L\}$ be a partition of $[a, b]$ so that $a \equiv \tau_0 < \cdots < \tau_i < \tau_{i+1} < \ldots < \tau_L \equiv b$ and define the *knot-vector* $\boldsymbol{\tau} \coloneqq [\tau_0, \ldots, \tau_L]$. Moreover, let $\gamma_i \coloneqq [\tau_i, \tau_{i+1})$ for $i = 0, \ldots, L-2$, $\gamma_{L-1} \coloneqq [\tau_{L-1}, \tau_L]$, choose $d \in \mathbb{N}$ a polynomial degree and set $k \coloneqq d + 1$ the corresponding polynomial order.

Definition 1. *The space of piecewise polynomial functions with degree d and knot-vector $\boldsymbol{\tau}$ is defined as follows*

$$P_{d,\boldsymbol{\tau}} \coloneqq \left\{ f : \Omega \to \mathbb{R} \mid f_{|\gamma_i} \in \Pi_d, \ \textit{for } i = 0, \ldots, L-1 \right\}, \tag{5}$$

where Π_d is the space of polynomials of degree less or equal to d.

$P_{d,\boldsymbol{\tau}}$ is a vector space on the field $\mathbb{R}$ and its dimension is $\dim\left(P_{d,\boldsymbol{\tau}}\right) = L\,(d+1)$.

Remark 1. *No regularity conditions are imposed in Definition 1, which implies that the elements of $P_{d,\boldsymbol{\tau}}$ may also not be continuous.*

In many CAGD applications, flexible tools are often desirable. Thereby, let $\boldsymbol{\mu} = \{\mu_1, \ldots, \mu_{L-1}\}$ the multiplicity vector with $1 \leq \mu_i \leq k$ for $i = 1, \ldots, L-1$ and define the *extended knot-vector*

$$\boldsymbol{t} \coloneqq [t_0, \ldots, t_{k-2}, t_{k-1}, \ldots, t_{n+1}, t_{n+2}, \ldots, t_{n+k}],$$

with

$$\boldsymbol{\tau} = [t_{k-1}, \ldots, t_{n+1}] = [\tau_0, \underbrace{\tau_1, \ldots, \tau_1}_{\mu_1}, \ldots, \underbrace{\tau_{L-1}, \ldots, \tau_{L-1}}_{\mu_{L-1}}, \tau_L].$$

Remark 2. *The knots $t_0, \ldots, t_{k-2}$ and $t_{n+2}, \ldots, t_{n+k}$ are the so called* auxiliary knots*; they can be chosen arbitrarily, and the only constraint they have to fulfill is to be ordered. A common choice consists in choosing them all coincident, respectively, to t_{k-1} and t_{n+1}, i. e. $t_0 \equiv \ldots \equiv t_{k-2} \equiv t_{k-1}$ and $t_{n+1} \equiv t_{n+2} \ldots \equiv t_{n+k-1}$. An extended knot vector with this auxiliary knot choice is said to be* open.

Definition 2. *The space of polynomial spline functions with degree d, global regularity $d - \max\{\boldsymbol{\mu}\}$ and knot-vector $\boldsymbol{t}$ is defined as follows*

$$V \coloneqq \left\{ f : \Omega \to \mathbb{R} \mid f^{(r)}(\tau_i^-) = f^{(r)}(\tau_i^+), \ \textit{for } r = 0, \ldots, d - \mu_i, \ i = 1, \ldots, L-1 \right\}. \tag{6}$$

V is a vector space on the field $\mathbb{R}$, $\Pi_d \subset V_{d,\boldsymbol{t}} \subset P_{d,\boldsymbol{\tau}}$ and its dimension is

$$n + 1 \coloneqq \dim V = L\,(d+1) - \sum_{i=1}^{L-1} (d - \mu_i + 1) = (m+1) + \sum_{i=1}^{L-1} \mu_i = k + \mu,$$

where $\mu = \sum_{i=1}^{L-1} \mu_i$.

The vector space V can be generated by $n+1$ univariate B-splines of degree d over $[a, b]$, which can be recursively defined [13, 11].

Definition 3. *Let $\beta_{j,k} : \Omega \to \mathbb{R}$ indicate the j-th B-spline of order $k \geq 1$, for each $j = 0, \ldots, n$. For $x \in \mathbb{R}$, if $k = 1$,*

$$\beta_{j,1}(x) = \begin{cases} 1, & \text{if } \ x \in [t_j, t_{j+1}), \\ 0, & \text{otherwise,} \end{cases}$$

and $\beta_{j,1}(t_{n+1}) = 1$. If $k \geq 2$,

$$\beta_{j,k}(x) = \omega_{j,k}(x)\beta_{j,k-1}(x) + (1 - \omega_{j+1,k}(x))\, \beta_{j+1,k-1}(x),$$

where $\omega_{j,k} : \mathbb{R} \to \mathbb{R}$ is a piecewise linear *polynomial of the form*

$$\omega_{j,k}(x) = \begin{cases} \frac{x - t_j}{t_{j+k-1} - t_j}, & \text{if } \ x < t_{j+k-1}, \\ 0, & \text{otherwise.} \end{cases}$$

Note that each B-spline $\beta_{j,k}$ for $j = 0, \ldots, n$ is a piecewise polynomial function of degree d, which has maximum local smoothness on each subinterval of the partition $\boldsymbol{t}$. On the other hand, the regularity at each unique knot $t_j \in \boldsymbol{t}$ is $d - \mu_j$, where $1 \leq \mu_j \leq k$ is the number of times the knot value t_j appears in $\boldsymbol{t}$. Moreover, the presence of repeated knots in $\boldsymbol{t}$ implies that choosing $t_{\hat{i}} \equiv t_{\hat{i}+1}$ for some $\hat{i}$ modifies the polynomial function $\omega_{\hat{i},\hat{r}}$ for some $\hat{r} \leq k$, which can result identically 0 and leads to a reduction of regularity of the B-splines $\beta_{\hat{i},r}$ for $r \geq \hat{r}$.

Univariate B-splines are characterized by three key properties, which follow directly from Definition 3, namely for each $j = 0, \ldots, n$,

1. non-negativity, i. e. $\beta_{j,k}(x) \geq 0$ for all $x \in \mathbb{R}$;
2. local support, i. e. $\beta_{j,k}(x) = 0$, if $x \notin [t_j, t_{j+k})$;
3. partition of unity, i. e. $\sum_{j=0}^{n} \beta_{j,k}(x) = 1$ if $x \in [t_{k-1}, t_{n+1}] \equiv [a, b]$.

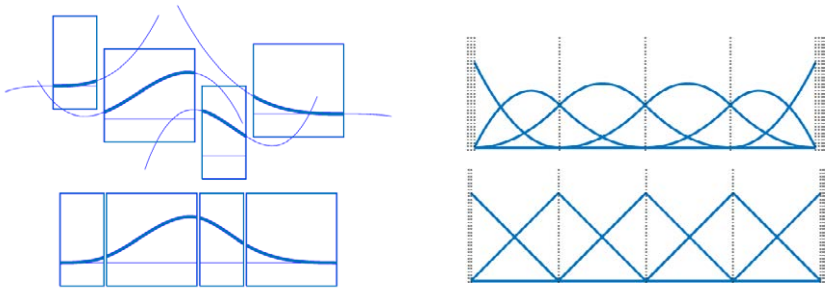
Finally, the $n + 1$ B-splines defined on the open knot vector $\boldsymbol{t}$ are piecewise polynomials of degree d; they belong to $C^{d-s}(\Omega)$, where $s \coloneqq \max_{1,\ldots,L-1} \mu_i$, and form a basis for the space V [30, 29], hence

$$V \coloneqq \operatorname{span}\{\beta_{0,k}, \ldots, \beta_{n,k}\}.$$

Figure 4 (a) illustrates a B-spline of degree $d = 3$ and its four polynomial pieces. Figure 4 (b) illustrates a B-spline basis of degree $d = 1$ (top) and a B-spline basis of degree $d = 2$ (bottom) on open knot-vectors $\boldsymbol{t}$ with uniform interior knots $\boldsymbol{\tau}$.

Remark 3. *Let $d \in \mathbb{N}$ and $k = d + 1$ be a polynomial degree and order respectively, and let $[0, 1] = \Omega \subset \mathbb{R}$. If a knot vector with no interior knots, i. e. $\boldsymbol{\tau} = [0, 1]$ and the corresponding extended* open *knot vector, i. e.*

$$\boldsymbol{t} = [\underbrace{0, \ldots, 0}_{k}, \underbrace{1, \ldots, 1}_{k}], \tag{7}$$



(a) (b)

Figura 4. (a) A B-spline of degree $d = 3$ and its decomposition in 4 polynomial pieces. (b) B-splines of degree $d = 1$ (top) and $d = 2$ (bottom) on a uniform open knot-vectors.

are considered, the B-spline basis in Definition 3 corresponds to the Bernstein polynomial basis on Ω *[110, 44], namely*

$$\beta_{j,k}(x) := \binom{k-1}{j} x^j (1-x)^{k-j-1}, \text{ for each } j = 0, \ldots, d. \tag{8}$$

Once the extended knot-vector $\boldsymbol{t}$, the degree d and the order k are set, any B-spline geometry $\boldsymbol{s} : \Omega \to \mathbb{R}^N$ can be represented as

$$\boldsymbol{s}(x) = \sum_{j=0}^{n} \boldsymbol{c}_j \beta_{j,k}(x), \quad \text{for} \quad x \in \Omega, \tag{9}$$

with coefficients $\boldsymbol{c}_j \in \mathbb{R}^N$, for $N \in \mathbb{N}_{\geq 1}$. In particular, for $N = 2$ and $N = 3$ the spline object $\boldsymbol{s}$ corresponds to a planar or spatial spline curve respectively. An illustrative example of these two cases is provided in Figure 5.

Remark 4. *By choosing the space configuration in Remark 3, the representation in* (9) *is called the Bernstein-Bézier form.*

For B-spline curves the following properties hold.

1. B-spline curves are invariant under affine transformations.
2. The value of a spline curve $\boldsymbol{s}$ at a site $x \in \Omega$ depends only on a convex combination of k coefficients $\boldsymbol{c}_j \in \mathbb{R}^N$. It follows that, if $x \in [t_i, t_{i+1}]$ for some $i = k-1, \ldots, n$,

$$\boldsymbol{s}(x) = \sum_{j=i-k+1}^{i} \boldsymbol{c}_j \beta_{j,k}(x).$$

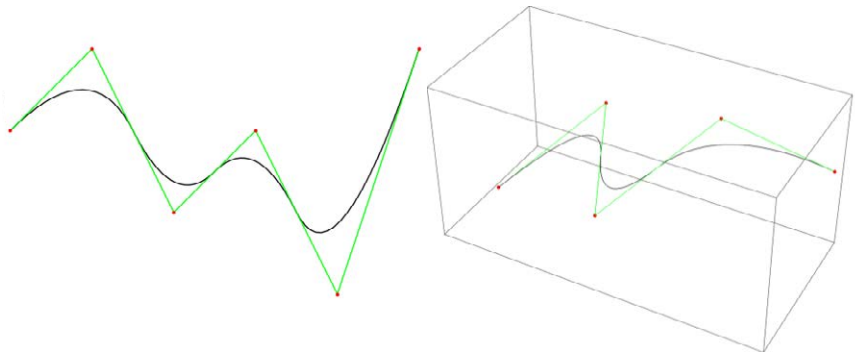


Figura 5. Planar (left) and spatial (right) B-spline curves.

3. Each segment of the B-spline curve belongs to the convex-hull of (only) k consecutive coefficients $\boldsymbol{c}_j \in \mathbb{R}^N$, hence it is usually addressed as *strong convex-hull*.

Remark 5. *The relationship between the value of the spline curve* $\boldsymbol{s}$ *and its coefficients* $\boldsymbol{c}_j$*, for* $j = 0, \ldots, n$ *has led to addressing them as* control points.

From these properties, it clearly follows that B-spline curves are extremely ductile, and consequently, they represent effective geometric modelling tools. Additional properties, e. g., *end-point interpolation* or *periodicity*, as well as algorithms, e. g., *knot-insertion* or *degree elevation*, to design and efficiently manipulate B-spline curves can be found in [28, 13, 44].

Tensor-product B-splines

Univariate B-splines can be easily extended to higher dimensions by considering a tensor-product construction. In particular, let Ω be a hypercube of $\mathbb{R}^D$, i. e.

$$\Omega \coloneqq \bigotimes_{h=1}^{D} [a_h, b_h], \text{ with } [a_h, b_h] \subset \mathbb{R} \text{ for } h = 1, \ldots, D.$$

Let $\boldsymbol{d} = (d_1, \ldots, d_D)$ be the polynomial multi-degree, $\boldsymbol{k} = (k_1, \ldots, k_D)$ be the corresponding polynomial multi-order and a suitable knot-vector in each parametric direction, i. e. $\boldsymbol{t}_1, \ldots, \boldsymbol{t}_D$. We define the tensor-product mesh G as $G \coloneqq \bigtimes_{h=1}^{D} \boldsymbol{t}_h$. A tensor-product B-spline $\beta_j : \Omega \subset \mathbb{R}^D \to \mathbb{R}$ is then defined as the tensor-product of D univariate B-splines,

$$\beta_{j,\boldsymbol{k}} \coloneqq \prod_{h=1}^{D} \beta_{j_h,k_h}, \text{ for } j_h = 0, \ldots, n_h, \ h = 1, \ldots, D.$$

More precisely, they can be all identified by a set of multi-indices, i. e.

$$\mathcal{B} = \{\beta_j \mid j \in \Gamma_{\boldsymbol{k}}\},$$

with

$$\Gamma_{\boldsymbol{k}} \coloneqq \{j = (j_1, \dots, j_D) \mid j_h = 1, \dots, n_h,\ h = 1, \dots, D\}. \tag{10}$$

Because of the tensor construction, many of the simple algebraic properties of univariate B-splines hold. In particular, non-negativity, locality, and partition of unity. Moreover, the tensor-product B-splines defined on the grid G are piecewise multivariate polynomials of multi-degree $\boldsymbol{d}$, their regularity is characterized by the multiplicity of the knot-lines of G and they form a basis for the tensor-product space, i. e.

$$V \coloneqq \operatorname{span}\{\mathcal{B}\}.$$

In Figure 6, we show an example of bivariate tensor product B-splines. In particular, in (a) we show the univariate B-spline basis of degree 3 (top) and 2 (bottom). In (b) we show 2 of the final $8 \times 9 = 72$ basis functions, namely $\beta_{4,4}\beta_{1,3}$ (top) and $\beta_{3,4}\beta_{5,3}$ (bottom). Finally, the bivariate tensor-product grid is shown in (c) and the resulting tensor-product basis is reported in (d).

Similarly to the univariate case, once a tensor-product grid G is chosen and both the multi-degree $\boldsymbol{d}$ and the multi-order $\boldsymbol{k}$ are set, any tensor-product B-spline geometry $\boldsymbol{s} : \Omega \subset \mathbb{R}^D \to \mathbb{R}^N$ can be represented as

$$\boldsymbol{s}(x) = \sum_{j \in \Gamma_{\boldsymbol{k}}} \boldsymbol{c}_j \beta_{j,\boldsymbol{k}}(x), \quad \text{for} \quad x \in \Omega, \tag{11}$$

with coefficients $\boldsymbol{c}_j \in \mathbb{R}^N$, for $N \in \mathbb{N}_{\geq 1}$. In particular, if $D = 2$, for $N = 2$ and $N = 3$ the spline geometry $\boldsymbol{s}$ corresponds to a planar or surface spline patch, respectively.

Remark 6. *Likewise in the univariate scenario, see Remark 3 and 4, let $\Omega = [0,1]^D$ and a tensor-product grid G built on* open *knot-vectors with no interior knots, i. e. $\boldsymbol{t}_h$ as in (7) for each $h = 1, \dots, D$. Then the tensor-product B-spline basis corresponds to the tensor-product Bernstein polynomial basis and the tensor-product B-spline geometries as in (11) are called tensor-product polynomial geometries in Bernstein-Bézier form.*

Also for the tensor-product multi-variate case, the properties addressed for B-spline curves hold, i. e. affine invariance, locality, and strong convex-hull. Therefore, tensor-product B-splines are quite flexible, and additionally, tensor-product B-spline basis have good numerical properties and are easy to evaluate [11, 144]. Note that the coefficients of a multivariate tensor-product B-spline geometry are called *control points*, as in Remark 5.

The main limitation of tensor-product B-spline structures arises when a refinement strategy occurs. More precisely, the space dimension tends to increase very fast when mesh refinement is considered, hence leading to

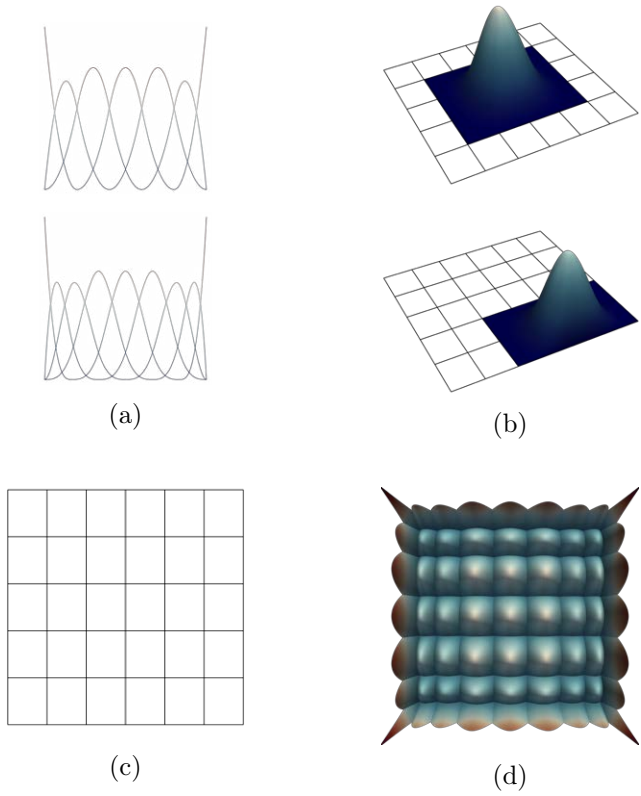


Figura 6. (a) Univariate B-spline basis of degree $d = 3$ on a uniform open knot-vector of 5 interior knots (top) and with degree $d = 2$ on a uniform open knot-vector with 4 interior knots (bottom). (b) Two example of the basis functions obtained by the tensor-product construction of the basis in (a). (c) The tensor-product grid G, built with the knot-vectors for the univariate basis in (a). (d) The final tensor-product basis of bi-degree $\boldsymbol{d} = (3, 2)$ on G.

computationally very expensive models. In particular, each time that a knot-line is inserted in one direction, the dimension of the tensor-product space increases a number of times equal to the product of the dimensions of the univariate spaces in all the other directions. Furthermore, when dealing with approximation problems, tensor-product structures are proven not to be flexible enough since they allow only global refinement guided by knot lines and preclude suitable adaptive and local refinement. Thereby, degrees of freedom are also added in regions far away from the area of interest. Moreover, these degrees of freedom in excess not only increase the computational costs in memory and time, but they are also unnecessary to improve the accuracy of the final model due to the local action of each control point. Figure 7 illustrates an example in the bivariate case of the non-locality of tensor-product refinement. In particular, a tensor-product

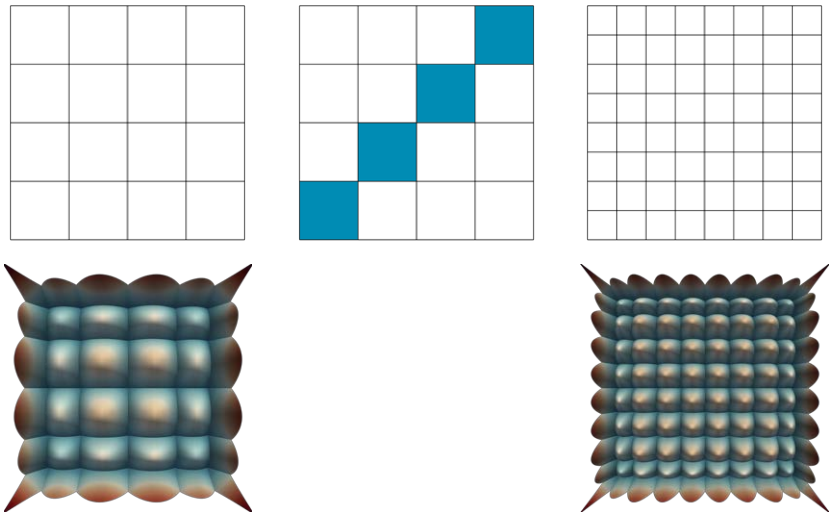


Figura 7. Tesor-product refinement: the initial mesh (top left) and basis (bottom left); the marked region of interest for refinement (top center); the final refined mesh (top right) and basis (bottom right).

grid (top left) is shown together with the corresponding tensor-product basis. Since the model needs to be refined, along a diagonal direction, the only possible way to refine the marked elements (top center) consists of performing a global uniform refinement, as shown in the resulting mesh (top right) and basis (bottom right).

THB-splines

When dealing with multivariate settings, the tensor-product constructions allow only a global distribution of the Degrees Of Freedom (DOFs) for the problem at hand, whereas local spline refinement enables the possibility of achieving flexible and accurate models by strongly reducing the total number of control points when compared with standard tensor-product B-spline representations. This type of local mesh refinement is intrinsically supported by hierarchical splines [53], where local refinement is achieved by introducing tensor-product B-splines on multiple hierarchical levels. A selection mechanism to properly identify a Hierarchical B-spline (HB-spline) basis was originally introduced in [87].

Let Ω be a hypercube of $\mathbb{R}^D$ and consider a nested sequence of L tensor-product B-spline spaces

$$V^0 \subset \ldots \subset V^{L-1}, \tag{12}$$

defined on $\Omega \subset \mathbb{R}^D$. More precisely for each level $\ell = 0, \ldots, L-1$, let $\boldsymbol{d}$ be a polynomial multi-degree and $\boldsymbol{k}$ is the corresponding multi-order. Moreover,

let $\beta_j^\ell : \Omega \to \mathbb{R}$ be the $j-$th tensor-product B-spline basis function of the space V^ℓ. Hence,

$$V^\ell = \operatorname{span}\left\{\beta_j^\ell \mid j \in \Gamma_{\boldsymbol{k}}^\ell\right\},$$

where $\Gamma_{\boldsymbol{k}}^\ell$ is the set of indices for the tensor-product B-spline basis of level ℓ, as defined in (10).

Remark 7. *As long as the tensor-product B-spline spaces are nested as in* (12)*, they can be characterized by different degrees/orders along the levels, see [160].*

In addition, each V^ℓ is associated with a rectilinear-grid G^ℓ and their (non-empty) quadrilateral elements q are commonly addressed as mesh cells of level ℓ. We also consider a nested sequence of closed domains

$$\Omega \equiv \Omega^0 \supset \ldots \supset \Omega^L = \emptyset,$$

so that each Ω^ℓ is the union of a subset of cells of G^ℓ, thus each domain boundary $\partial\Omega^\ell$ is aligned with the knot lines of V^ℓ, for each $\ell = 0, \ldots, L-1$. The hierarchical mesh is defined as

$$\mathcal{M} \coloneqq \left\{q \in \mathbb{G}^\ell \mid \ell = 0, \ldots, L-1\right\},$$

where for each $\ell = 0, \ldots, L-1$,

$$\mathbb{G}^\ell \coloneqq \left\{q \in G^\ell \,|\, q \subset \Omega^\ell \setminus \Omega^{\ell+1}\right\}$$

is called the set of *active cells* of level ℓ.

Remark 8. *Note that the definition of a rectilinear-grid G or a hierarchical mesh $\mathcal{M}$ over the domain Ω induces a domain tessellation $T(\Omega)$, as considered in Chapter 2.*

The hierarchical spline construction consists in replacing any B-spline of level ℓ with support completely contained in $\Omega^{\ell+1}$ by B-splines at successively refined levels.

Definition 4. *The HB-spline basis is defined as follows*

$$\mathcal{H} \coloneqq \left\{\beta_j^\ell \mid j \in A_{\boldsymbol{k}}^\ell,\ \ell = 0, \ldots, L-1\right\} \tag{13}$$

where

$$A_{\boldsymbol{k}}^\ell \coloneqq \left\{j \in \Gamma_{\boldsymbol{k}}^\ell \mid \mathit{supp}\left(\beta_j^\ell\right) \subseteq \Omega^\ell \wedge \mathit{supp}\left(\beta_j^\ell\right) \not\subseteq \Omega^{\ell+1}\right\} \tag{14}$$

is the set of indices of active functions *and $\mathit{supp}\left(\beta_j^\ell\right)$ denotes the intersection of the support of β_j^ℓ with Ω^0. The corresponding hierarchical space is defined as*

$$V \coloneqq \mathit{span}\left\{\mathcal{H}\right\}.$$

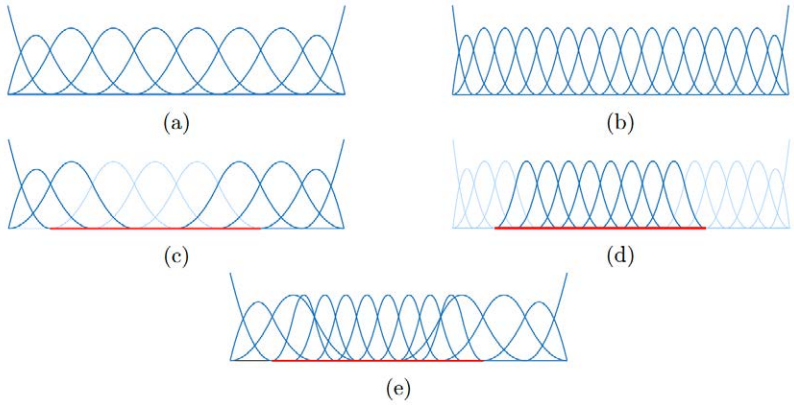


Figura 8. Example of univariate HB-spline basis of degree 2: (a) B-spline basis for V^0; (b) B-spline basis for V^1; (c) the active B-splines of level 0; (d) the active B-splines of level 1; (e) the HB-spline basis. The refinement area Ω^1 is also shown (red line).

A univariate example of HB-spline space with 2 level of refinement is provided in Figure 8, where two *nested* univariate B-spline basis of degree $d = 2$ are defined on a domain $\Omega \subset \mathbb{R}$. More specifically, the basis displayed in (a) defined on an open uniform knot-vector with 7 interior knots, whereas the basis displayed in (b) it is built on an open uniform knot-vector with 16 interior knots, obtained by dyadically splitting each non-empty knot span of the coarse knot-vector. The subdomain Ω^1 is highlight in red in (c), (d) and (e), where $\Omega^1 \subset \Omega^0 \equiv \Omega$. In particular, (c) and (d) show the active univariate B-splines at level $\ell = 0$ and $\ell = 1$, respectively, according to the selection mechanism (14). Finally, (d) illustrates the 2 level HB-spline basis, according to Definition 4 and (14).

The hierarchical structure enables to localize the refinement only in the area of the domain where it is needed. The difference between bivariate tensor-product refinement and hierarchical dyadic mesh refinement on the same area of interest, as well the the respective basis, can be visualized in Figure 9. In particular, we show the refined mesh and the corresponding basis of bi-degree $\boldsymbol{d} = (2, 2)$, obtained by refining the same area of interest, up to 3 hierarchical levels. It is evident that in the hierarchical approach the refinement is local and adds less basis elements. In this case, a dyadic refinement procedure is employed, that is a single element is subdivided into four smaller elements. Again, the small element can be refined as well, so that we get a refined grid over several levels. Note that the refined tensor-product basis has 729 basis functions, whereas the refined hierarchical basis functions has 184 basis functions.

Directly from Definition 4 and from tensor-product B-spline properties, it immediately follows that HB-splines have a local support and are non-negative. Moreover, linear independence can be achieved with the help of the local linear independence of the spline basis at each hierarchical level

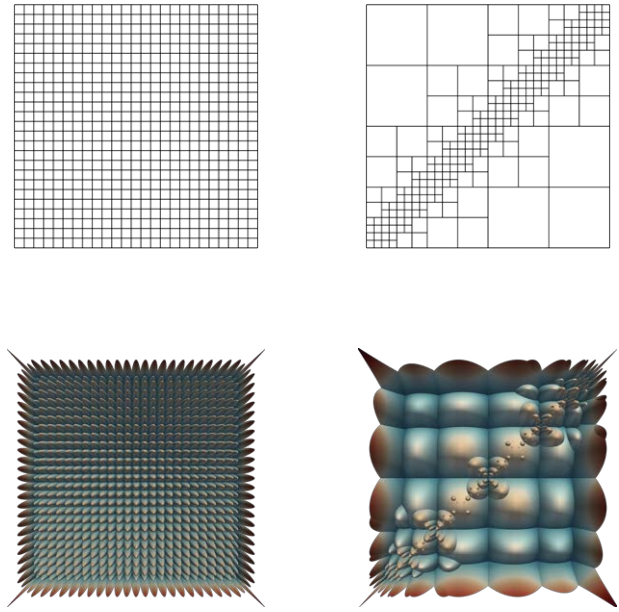


Figura 9. Comparison between tensor-product (left) and hierarchical refinement on 3 levels (right): the tensor-product mesh (top left) with the corresponding basis (bottom left) and the hierarchical mesh (top right) with the hierarchical basis (bottom right).

[87, 160].

On the other hand, HB-spline basis are characterized by different overlaps of coarse and fine B-spline basis function supports, and the partition of unity property of the basis is lost. A simple way to recover the partition of unity property consists in suitably weighting the basis, by applying a adequate scaling factor to the HB-spline basis functions [160]. Nevertheless, there is no guarantee that the weights would be positive or non-zero, and for some HB-spline configuration a proper scaling is not guaranteed to exists at all. In order to ensure the existence of a scaling and the positivity of the weights, additional constraints on the subdomain configurations need to be considered [160].

This motivates the construction of another basis for the hierarchical spline space, the Truncated Hierarchical B-spline (THB-spline) basis [55]. The THB-spline basis maintains the partition of unity property without any additional assumption on the considered subdomains. In addition to the partition of unity, THB-spline basis provides various useful properties in the mathematical framework of hierarchical splines, such as strong stability, full approximation power, and efficient implementation [84, 149]. In particular, a THB-spline basis can be obtained from a HB-spline basis, by applying a truncation mechanism to each HB-spline basis function. In particular, the truncation mechanism preserves all the properties of HB-spline, such as linear independence and non-negativity.

For any $\ell = 0, \dots, L-2$, let $s \in V^\ell \subset V^{\ell+1}$ be a tensor-product spline represented in terms of the basis of the refined space $V^{\ell+1}$ as

$$s(\boldsymbol{x}) = \sum_{j \in \Gamma_{\boldsymbol{k}}^{\ell+1}} c_j^{\ell+1}(s) \beta_j^{\ell+1}(\boldsymbol{x}), \quad \text{for } \boldsymbol{x} \in \Omega,$$

for suitable coefficients $c_j^{\ell+1}(s)$, for each $j \in \Gamma_{\boldsymbol{k}}^{\ell+1}$. By separating the basis functions of $V^{\ell+1}$ whose supports are not completely contained in $\Omega^{\ell+1}$, from the remaining ones, it holds,

$$\begin{aligned} s(\boldsymbol{x}) = & \sum_{\substack{j \in \Gamma_{\boldsymbol{k}}^{\ell+1} \\ \operatorname{supp}\left(\beta_j^{\ell+1}\right) \subseteq \Omega^{\ell+1}}} c_j^{\ell+1}(s) \beta_j^{\ell+1}(\boldsymbol{x}) + \\ & \sum_{\substack{j \in \Gamma_{\boldsymbol{k}}^{\ell+1} \\ \operatorname{supp}\left(\beta_j^{\ell+1}\right) \not\subseteq \Omega^{\ell+1}}} c_j^{\ell+1}(s) \beta_j^{\ell+1}(\boldsymbol{x}), \text{ for } \boldsymbol{x} \in \Omega. \end{aligned}$$

Definition 5. *The truncation of $s \in V^\ell$ at level $\ell+1$ is defined as*

$$\operatorname{trunc}^{\ell+1}(s) \coloneqq \sum_{\substack{j \in \Gamma_{\boldsymbol{k}}^{\ell+1} \\ supp\left(\beta_j^{\ell}\right) \not\subseteq \Omega^{\ell+1}}} c_j^{\ell+1}(s)\, \beta_j^{\ell+1} \tag{15}$$

and the cumulative truncation with respect to all finer levels is

$$\operatorname{Trunc}^{\ell+1}(s) \coloneqq \operatorname{trunc}^{L-1}\left(\operatorname{trunc}^{L-2}\left(\dots\left(\operatorname{trunc}^{\ell+1}(s)\right)\dots\right)\right), \tag{16}$$

with $\operatorname{Trunc}^L(s) \equiv s$, for $s \in V^{L-1}$. Finally, the THB-spline basis for the hierarchical space can be defined as

$$\mathcal{T} \coloneqq \left\{\tau_j^\ell = \operatorname{Trunc}^{\ell+1}\left(\beta_j^\ell\right) \mid j \in A_{\boldsymbol{k}}^\ell,\, \ell = 0, \dots, L-1\right\}, \tag{17}$$

where the B-spline β_j^ℓ is the mother *B-spline of the truncated B-spline τ_j^ℓ.*

Figure 10 shows an example of the truncation mechanism for univariate B-splines over 2 hierarchical levels, for $\ell = 0, 1$. More precisely, the coarse B-spline (dashed) belonging to V^0 can be represented in terms of the 5 finer B-splines (in the background) belonging to V^1, for suitable coefficients $c_0, \dots, c_4$. In addition, we choose a subdomain Ω^1, highlighted in red in the picture. The truncation of the coarser B-spline corresponds to setting to 0 the coefficients of the finer B-splines whose supports are fully contained in Ω^1, i. e. the last two finer B-splines on the right of the picture.

THB-splines are non-negative, have local support, form a partition of unity, and generate the same space of the HB-spline basis [55], i. e.

$$\operatorname{span}\{\mathcal{H}\} \equiv \operatorname{span}\{\mathcal{T}\}. \tag{18}$$

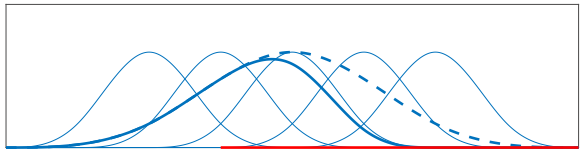


Figura 10. Truncation mechanism defined in (15) for a B-spline of degree 4. The B-spline of level $\ell = 0$ (dashed line) is truncated over the subdomain Ω^1 (in red) via 5 finer B-splines (thin lines) for $\ell = 1$ and results in a THB-spline (thick line).

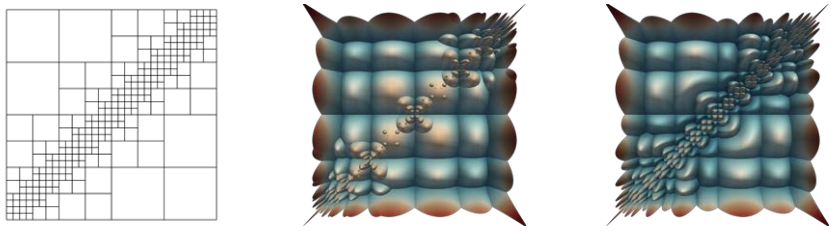


Figura 11. Comparison between hierarchical bases: the hierarchical mesh (left) and the corresponding HB-spline (center) and THB-spline (right) bases.

The effectiveness of employing THB-splines both for geometric design and isogeometric analysis has been shown in [62], among others. Figure 11 shows the comparison, for the same hierarchical space characterized by the hierarchical mesh on the left, of the HB-spline basis and THB-spline basis, in the center and on the right of the picture, respectively. Note that the THB-spline basis functions reduce the overlaps of the supports related to THB-splines introduced at different hierarchical levels.

Because of their properties, THB-splines are a desirable tool for building flexible geometric models. They have been here presented in their more general form, where their construction can be developed in any parametric and physical dimension, namely for any $D, N \in \mathbb{N}_{\geq 1}$. In particular, a (TH)B-spline geometry is a linear combination of (TH)B-splines, defined as

$$\boldsymbol{s}(\boldsymbol{x}) = \sum_{\ell=0}^{L-1} \sum_{j \in A_{\boldsymbol{k}}^{\ell}} \boldsymbol{c}_j \tau_j^{\ell}(\boldsymbol{x}), \quad \boldsymbol{x} \in \Omega, \tag{19}$$

with $\boldsymbol{c}_j \in \mathbb{R}^N$ for $j \in A_{\boldsymbol{k}}^{\ell}$, $\ell = 0, \ldots, L-1$. For $D = 1$ a planar ($N = 2$) or spatial ($N = 3$) (TH)B-spline curve can be specified, whereas for $D = 2$ and $N = 3$ a THB-spline surface can be constructed.

1.2. Deep learning with neural networks

For centuries science has been an activity operated by humans for humans; nevertheless, with the advent of programmable computers, a new perspective has been developed and still subsists, in which science becomes

an activity performed by humans and machines for humans and machines. The paradigm of using computers to simulate human intelligence was first described in [156], where the "Turing test" was proposed to determine whether computers were capable of human intelligence. Subsequently, the term Artificial Intelligence (AI) was coined [117] as the science and engineering of developing non-biological systems that mimic human behaviour and intelligence. AI began as a simple series of *if statements* and has developed over several decades, resulting in more advanced algorithms that perform similarly to the human brain. Nowadays, it is a flourishing field with many applications and active research topics. We exploit intelligent software to automate routine labour, understand speeches or images, make diagnoses in medicine, and support scientific research. Furthermore, the recent increase in data availability led to the development of new AI techniques capable of handling large data volumes. In this respect, Machine Learning (ML) is the field of AI consisting of algorithms based on numerical and statistical methods that directly *learn* from data without being explicitly programmed [141, 120]. More specifically, throughout this Thesis we will consider the following concept of learning.

Definition 6 ([120])**.** *A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at task in T, as measured by P improves with experience E.*

Consequently, any learning model should be able to properly generalize based on its own experience. The core of any learning problem are *data*, which we assume to come in a finite amount, hence this results in an intrinsically difficult problem as we have to generalize from limited data information.

The data employed to define ML models are usually separated into different *pairwise disjoint* sets, which play a role at different stages of the creation of the models, i. e. *training*, *validation* and *test* sets. More precisely, the training dataset is a set of data used to tune the parameters of the ML model during the so-called *training* phase. Many methodologies that search through training data for empirical relationships often may happen to overfit the data, meaning that they can identify and exploit apparent relationships in the training data that lack general applicability. For this reason, a validation data set is often used during the training phase to analyze an unbiased evaluation of the model and implement the early stopping of the training phase. Finally, the test data set is used to provide an unbiased evaluation of the *final* trained model and analyze its generalization capacity.

The nature of the data plays a role also for the definition of the learning paradigm. In particular, we recall the following learning types, among others.

1. *Supervised Learning*: in this case, the data comprises examples of the input items, along with their corresponding target. The goal consists in building a function that maps (new) data on expected target values. Examples of its applications are email spam detection or hand-

written digits recognition. In general, this is the most common scenario associated with classification, regression, and ranking problems.

2. *Unsupervised learning*: in this case, all the data is unlabelled. On the other hand, usually the data contains many features which are therefore exploited to learn useful properties of the structure of this data. Since in general no labelled example is available in this setting, it can be difficult to quantitatively evaluate the performance of a model. Clustering and dimensionality reduction are example of unsupervised learning problems.

The contributions of this Thesis related to learning problems mainly act within the variety of *unsupervised learning* framework. In order to provide a rough idea of the learning approaches, we list additional existing learning paradigms, which we do not tackle in this Thesis, i. e. Semi-supervised learning [23], Transductive inference [157], Online learning [140] and Reinforcement learning [154]. In practice, many more intermediate learning scenarios may been encountered, see [6] for more details.

Given two measurable spaces $\mathcal{P}$ and $\mathcal{X}$, the final goal of DL models consists in approximating a certain function $f : \mathcal{P} \to \mathcal{X}$. The workhorse of DL are NNs, i. e. computational models that are composed of several processing layers to learn representations of data, with multiple levels of abstraction. Therefore, a NN approximates f with a parametric mapping $x = \phi(\boldsymbol{p}; \boldsymbol{\omega})$ by automatically performing a data feature extraction process, which discover *intrinsic* relations in large data sets, and using backpropagation algorithms [138] to determine how suitably change the internal parameters $\boldsymbol{\omega}$ to result in the best function approximation of f. More precisely, throughout this Thesis we will consider as DL models, only the ones which agree with the following definitions.

Definition 7 ([64])**.** *The term NN indicates a collection of functions that can be represented by Directed Acyclic Graphs (DAGs), where the vertices correspond to elementary almost everywhere differentiable parametric functions and the edges symbolize compositions of these functions. The vertices of these DAGs are usually called* nodes *or* units*.*

Definition 8 ([4])**.** *DL refers to techniques where deep NNs are used to define the model and their parameters are set with gradient-based methods.*

This latter definition synthesize the two essential cores of DL technologies, i. e. deep NNs and gradient-based optimization, which will be both subsequently addressed in this Chapter.

Different NN architectures determine different DL models, where the word *architecture* refers to the overall structure of the network: how many units it has and how these units are connected to each other. The amount of different possible NN architecture topologies is nowadays incredible, thereby in this chapter we provide insights of the first architecture ever developed, i. e. MLP and for then focusing on CNNs and GCNs, employed later on in this Thesis.

Multi Layer Perceptron

The most basic deep NN is the MLP, which has its foundations in the Perceptron [118, 137], i. e. an algorithm developed to solve the binary classification problem. More precisely, the problem to be solved can be formalised as in the following definition.

Definition 9. *Let $\mathcal{P}, \mathcal{X}$ be two measurable spaces and let*

$$\{(\boldsymbol{P}_i, x_i) \in \mathcal{P} \times \mathcal{X} \ : \ i = 1, \dots, m\}$$

be a dataset consisting of input features $\boldsymbol{P}_i \in \mathcal{P}$ and corresponding labels $x_i \in \mathcal{X}$. A binary-classification task consists in finding a model $f : \mathcal{P} \to \mathcal{X}$, with $\mathcal{X} = \{-1, +1\}$, so that $f(\boldsymbol{P}_i)$ is a good prediction of the true label x_i, for $i = 1, \dots, m$.

The Perceptron defines a two-class model, which maps a real-valued input $\boldsymbol{P}_i$ into $x_i \in \{-1, +1\}$, by using a non-linear *activation* function σ. Its architecture contains a single input layer and an output node, respectively in green and red in Figure 12. The input layer contains m input units that transmit the features $\boldsymbol{P}_i = [p_1, \dots, p_m]$ with edges of weight $\boldsymbol{\omega} = [\omega_1, \dots, \omega_m]$ and *bias* b, to the output unit x. More precisely, the linear function

$$\boldsymbol{\omega} \cdot \boldsymbol{P}_i \coloneqq \sum_{j=1}^{m} \omega_j p_j + b \tag{20}$$

is computed at the output node and subsequently, the activation function σ is applied to this real value in order to predict the dependent variable of $\boldsymbol{P}_i$. Specifically, for the Perceptron model, σ corresponds to the function $\operatorname{sign} : \mathbb{R} \to \{-1, +1\}$, which maps a real value z to either $+1$ or -1, i. e.

$$\operatorname{sign}(z) = \begin{cases} +1 \text{ if } z \geq 0, \\ -1 \text{ if } z < 0, \end{cases}$$

hence resulting naturally appropriate for binary classification. Finally, the prediction x is computed as

$$x = \sigma\left(\sum_{j=1}^{m} \omega_j p_j + b\right) = \operatorname{sign}\left(\sum_{j=1}^{m} \omega_j p_j + b\right). \tag{21}$$

The error of the prediction is consequently a value drawn from the set $\{-2, 0 + 2\}$. In the case the error is non-zero, the weight values $\boldsymbol{\omega}$ and the bias b of the Perceptron need to be updated along the (negative) direction of the error gradient, to recover and avoid misclassification.

Putting together more Perceptron architectures and different activation functions results in a MLP model. For architectures with *multiple layers*, these are classified as *input*, *hidden* and *output* layers, depending on the topology of their connections. More precisely,

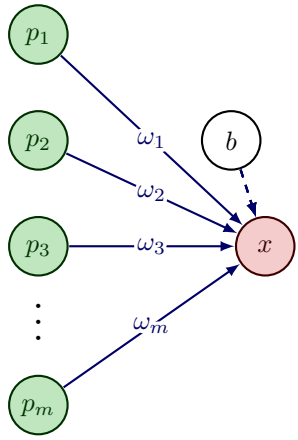


Figura 12. Example of a Perceptron with m input units and 1 output unit.

- *Input layer*: in this case, each unit corresponds to a feature of the input sample; hence, there are as many units as the number of features of the considered data. Moreover, the direct graph connections are only in output, from the units of the input layers towards the units of the following one.
- *Hidden layers*: these layers are placed between the input and the output layers. Their number depends on the chosen architecture, and their amount of units is unconstrained and can vary within the layers. The number of hidden layers and hidden units per layer plays a role in the expressiveness of the model. Finally, any hidden unit can have connections in input and output, respectively, with the units of the previous or following layer.
- *Output layer*: the number of units of the output layers depends strictly on the function at hand, which the NN is required to approximate. In this case, there is no next layer, and the output unit connections are only along the incoming direction.

Figure 13 shows an MLP with 5 layers, i. e. an input layer with m units (green), 3 hidden layers with m_1, m_2, m_3 units per layer (blue) and 1 output layer with n units (red). In particular, for an MLP architecture, at any hidden layer, each unit receives the same real value from every unit of the previous layer and produces a real value that is passed in output to every unit of the following layer. Due to their edge topology, MLP architectures are also addressed as Fully Connected NN.

Definition 10 ([4])**.** *Let $L \in \mathbb{N}_{\geq 1}$, where $L+1$ is the total number of layers of a MLP architecture. For each $\ell = 0, \ldots, L$, let m_ℓ be the number of hidden units for the ℓ-th layer and let*

$$\sigma_\ell : \mathbb{R} \to \mathbb{R}$$

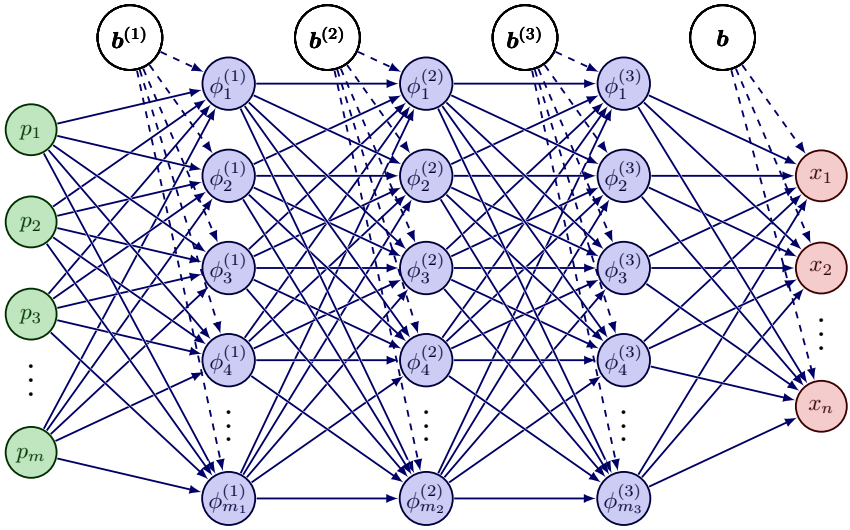


Figura 13. Example of an MLP with 5-layer Perceptron, m input units, m_ℓ hidden units for each ℓ^{th} hidden layer and n output units.

be an activation function. Moreover, let

$$W = \left\{W^{(\ell)}\right\}_{\ell=1}^{L} \quad \textit{and} \quad b = \left\{\boldsymbol{b}^{(\ell)}\right\}_{\ell=1}^{L},$$

where

$$W^{(\ell)} = \left(w_{ij}^{(\ell)}\right)_{i=1,j=1}^{m_\ell, m_{\ell-1}} \in \mathbb{R}^{m_\ell \times m_{\ell-1}} \quad \textit{and} \quad \boldsymbol{b}^{(\ell)} \in \mathbb{R}^{m_\ell}, \quad \textit{for} \quad \ell = 1, \dots, L$$

are the weight matrices, and the bias vectors of the architecture, respectively. More precisely, $w_{ij}^{(\ell)}$ is the weight associated with the edge between the i-th unit of the ℓ-th layer and the j-th unit of the $\ell-1$ layer, whereas $b_i^{(\ell)}$ is the bias associated with the i-th unit of the ℓ-th layer, for $\ell = 1, \dots, L-1$.

Finally, let $\#LP$ the total number of learnable parameters of the architecture. The MLP realization function $\Phi : \mathbb{R}^{m_0} \times \mathbb{R}^{\#LP} \to \mathbb{R}^{m_L}$ can be defined as

$$\Phi\left(p, \{W, b\}\right) = \phi^{(L)}\left(p, \{W, b\}\right),$$

where

$$\begin{aligned}
&\phi^{(1)}\left(p, \{W, b\}\right) = W^{(1)} p + b^{(1)}, \\
&\hat{\phi}^{(\ell)}\left(p, \{W, b\}\right) = \sigma_\ell\left(\phi^{(\ell)}\left(p, \{W, b\}\right)\right) \quad \textit{for} \quad \ell = 1, \dots, L-1, \\
&\phi^{(\ell+1)}\left(p, \{W, b\}\right) = W^{(\ell+1)} \hat{\phi}^{(\ell)}\left(p, \{W, b\}\right) + \boldsymbol{b}^{(\ell+1)} \quad \textit{for} \quad \ell = 1, \dots, L-1.
\end{aligned}$$

The underlying directed acyclic graph of any MLP model is given by the composition of the affine linear maps $\phi| \to W^{(\ell)}\phi + \boldsymbol{b}^{(\ell)}$ $\ell = 0, \dots, L$, with

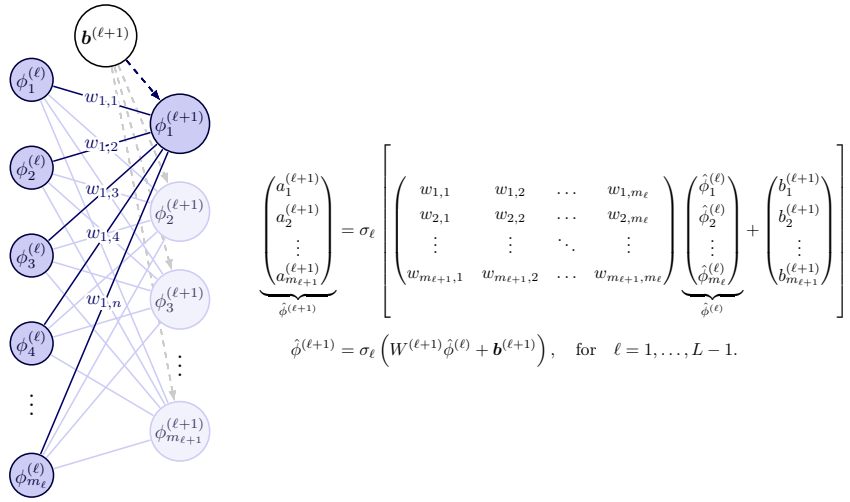


Figura 14. Example of MLP architecture realization, according to Definition 10.

the activation function σ_ℓ intertwined, for $\ell = 1, \dots, L-1$. An example of an MLP realization at general layer $\ell + 1$ is illustrated in Figure 14.

The choice of activation functions is a critical part of neural network design, and it depends on the problem at hand. In general, they are needed to be non-linear functions in order to avoid the trivial situation in which the output is only a linear combination of the input data [64]. In addition, their concurrence allows to represent arbitrarily complex functions [1]. A selection of common activation functions is presented in Figure 15.

Remark 9. *The Rectified Linear Unit activation function (ReLU) activation function is commonly used for its simple piecewise linear structure and high performance [168].*

The overall number of layers $L + 1$ gives the *depth* of the model. The name "deep learning" arose from this terminology. By adding more layers and more units within a layer, a deep NN can represent functions of increasing complexity, but of course the computational costs increase. It is fundamental to find a proper architecture that is a trade-off between computational training costs and the accuracy that can be achieved.

Remark 10. *With the notation introduced in Definition 10, the total number of learnable parameters $\#LP$ of a MLP architecture can be computed as,*

$$\#LP := \sum_{\ell=1}^{L} m_\ell \left(m_{\ell-1} + 1 \right).$$

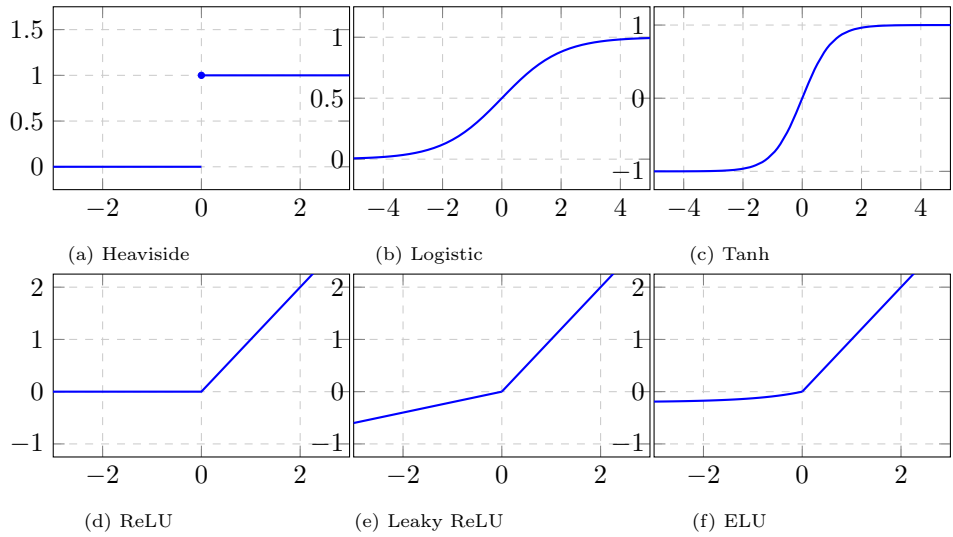


Figura 15. Some common activation functions, $\sigma(x)$ for $x \in \mathbb{R}$.

Convolutional Neural Networks

The development of CNN arose with the application of deep NN to image processing [92, 93]. More precisely, fully-connected networks ignore a key property of images, which is that nearby pixels are more strongly correlated than the distant ones. Many of the modern approaches to computer vision exploit this property by extracting *local* features that depend only on small subregions of the image. Subsequently, the information extracted from such features can then be merged into later processing stages.

In order to achieve more sparse learning models, the matrix multiplication operator, which characterized fully-connected architectures outlined in Definition 10, has been substituted by a matrix convolution operator [64]. More precisely,

Definition 11 ([64])**.** *CNNs are NNs that use convolution in place of general matrix multiplication in at least one of their layer.*

The general formulation of two matrix convolution $A, W \in \mathbb{R}^{m\times n}$ is

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mm} \end{pmatrix} \star \begin{pmatrix} \omega_{11} & \omega_{12} & \dots & \omega_{1n} \\ \omega_{21} & \omega_{22} & \dots & \omega_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \omega_{m1} & \omega_{m2} & \dots & \omega_{mm} \end{pmatrix} = \sum_{i=0}^{m-1}\sum_{j=0}^{n-1} a_{m-i,n-j}\omega_{1+i,1+j}. \tag{22}$$

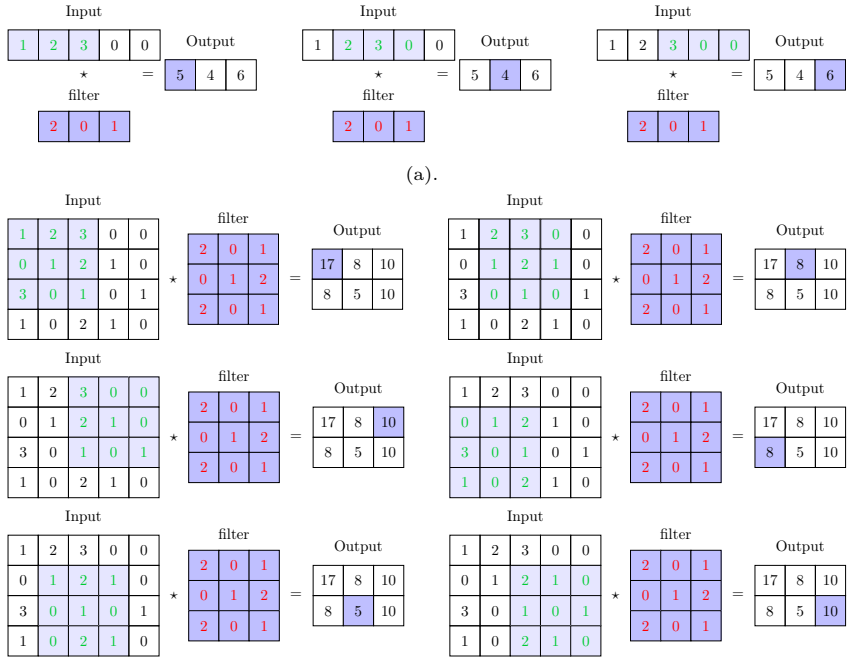


Figura 16. Examples of convolutional operators for $1D$ (a) and $2D$ (b).

Figure 16 (a) shows an example of $1D$ (univariate) convolution, whereas Figure 16 (b) shows and example of $2D$ (bivariate) convolution. In both cases, the input is processed by a filter through a matrix convolution operator, according to (22), which generates the output feature map, usually given in input to the successive convolutional operator. In particular, each output item is given by the sum of the element-wise product between the different *local* values of the input (green numbers in Figure 16) and the corresponding values of the filter (red numbers in Figure 16). Subsequently, the filter matrix slides on the input matrix, from left to right in the picture, until the input elements have all been completely scanned.

Thanks to the nature of convolutional operators, CNNs are characterized by three unique properties: sparse connectivity, weight sharing, and shift equivariant representations [6, 64]. Sparse connectivity means that the networks have *local* receptive fields, which collect information jointly from spatially neighbouring inputs. Weight sharing is achieved by requiring the receptive field to assume the same values on different instances of the input, and, consequently, the same parameters are involved to compute each output unit. Finally, shift equivariant representations follow the definition of the convolutional operations and the previous two properties. The locality and weight sharing properties for the one ($1D$) and the two dimensional ($2D$) convolutional operators can be seen again in Figure 16.

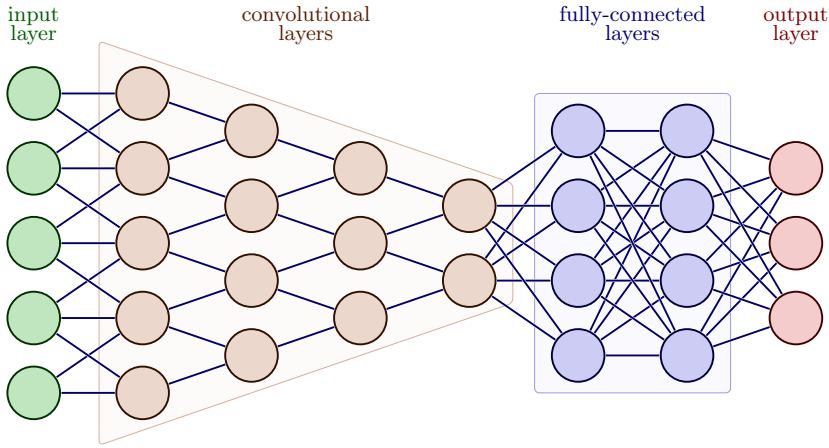


Figura 17. Example of a CNN.

Finally, Figure 17 shows an example of CNN. More precisely, it represents a deep NN characterized by 1 input layer (green), 6 hidden layers, i. e. 4 *convolutional* layers (orange), and 2 fully connected layer (blue), and 1 output layer (red). Note that the connections within the convolutional layers are less than the connections between the fully connected layers.

CNNs have been very successful in practical applications for processing data with a know *grid-like* topology [64, 88, 151, 162]. Examples include time-series data, which can be thought of as a $1D$ grid taking samples at regular time intervals, and image data,which can be thought of as a $2D$ grid of pixels.

Graph Convolutional Neural networks

In many applications, the data does not exhibit a grid-like structure but may be characterized by a graph or mesh structure or be completely unorganized, as in the case of scattered data. In order to handle this kind of data, a new learning theory and related architectures have been developed.

Geometric deep learning refers to the generalization of convolutional neural networks to non-Euclidean data such as discrete manifolds, graphs and general point clouds [19]. While for data represented on a regular grid in a Euclidean space the convolution operator is uniquely defined for arbitrary dimensions, in the case of unstructured non-Euclidean data many different approaches for the definition of operators that mimic the behaviour of standard convolution have been proposed [167].

More precisely, GCN are the CNN counterparts for non-gridded-like domains. There are two approaches to designing convolutional filters on graphs, namely the *spatial* and *spectral* methods, which result in the definition of *spatial*-GCNs and *spectral*-GCNs, see again [167].

Spatial methods define graph convolutions based on spatial relations between graph vertices in a similar fashion as standard CNNs. The updated

output vertex is obtained by its convolution with certain neighbours by means of a filter kernel that transfers the information along the neighbourhood edges; see [65, 121] as examples of spatial GCNs. Even if the definition of a spatial graph convolutional operator seems to be a natural extension of standard convolutions, it suffers from the issue of matching local neighbourhoods and uniquely defining translations on graphs from a spatial perspective [20].

Spectral-based methods have their foundations in the spectral graph theory, which links the discrete setting to the continuous one [27]. In particular, the convolution theorem defines convolutions as linear operators by exploiting the Fourier basis represented in terms of eigenvectors of the Laplace operator. With suitable choices of filter parameterization, the filters defined in the spectral domain are naturally localized, and the costs arising from computing the Fourier transform can be contained [33]. Additional examples of spectral GCNs are [20, 98], among others. A comprehensive survey of graph convolutional operators can be found in [167].

Optimization

We compute an objective function that measures the error (or distance) between the output and the desired target. The machine then modifies its internal, adjustable parameters to reduce this error. In a typical deep-learning system, there may be hundreds of millions of these adjustable weights and hundreds of millions of (labeled) examples with which to train the machine.

The second ingredient, gradient-based optimization, is made possible by the observation that, due to the graph-based structure of any neural network, see e. g., Definition 10, the gradient of an objective function with respect to the parameters of the neural network can be computed efficiently. This has been observed in various ways; see [82, 38, 138].