UNIVERSITAT JAUME I

Compitational Mathematics Degree

Final Degree Project

Geometric foundations for Geometry Processing of Neural Implicit Representations of Signed Distance Functions

Author:
Xavier Anadón García-Arquimbau

Academic tutor:
José Joaquín Gual Arnau

Presentation Date: 9 of June 2023
Curso académico 2022/2023

Resumen

This work explores using neural networks to approximate Signed Distance Functions for representing 3D shapes. The work explains fundamental geometry concepts and their application to these functions, as well as the workings of neural networks as function approximators. By connecting these concepts, it explains how neural networks can represent 3D shapes and how to perform shape smoothing and sharpening on them. The objective of this work was to provide a solid foundation for further exploration of this topic, with a focus on providing a mathematical explanation of these techniques. Future research can explore other types of operations and efficient ways of creating and manipulating these representations.

Keywords

Neural Networks, Geometry, Signed Distance Functions, Curvature.

Contents

Chapter 1

Introduction

1.1 Context and motivation

In recent years, deep learning has emerged as a rapidly growing area of research. While it initially gained popularity for its success in image processing, it has since been applied successfully to other modalities such as text, audio, and video. More recently, researchers have attempted to apply deep learning techniques to 3D data, including meshes and point clouds. This has the potential to improve our understanding of 3D environments and enable the creation of 3D generative models that have applications in fields such as robotics, virtual reality, and augmented reality.

One particularly interesting research direction involves using deep learning to represent 3D scenes and objects through the use of neural fields. These fields have numerous applications, as outlined in [16], including the approximation of Signed Distance Functions (SDFs) that represent a 3D shape. Throughout this text, we will provide a detailed explanation of these terms.

However, in order for these representations to be widely adopted, it is necessary to be able to perform geometry processing operations on them, such as shape smoothing and sharpening. Unfortunately, it is not a trivial task to perform these operations on SDFs represented using neural networks.

1.2 Objectives

The objective of this work is to provide a foundational understanding of how 3D shapes can be represented using Signed Distance Functions approximated by neural networks. Additionally, we will explain how geometry processing operations can be performed on these representations, specifically focusing on the example of shape filtering (i.e., smoothing and sharpening). Our work is strongly based on the research of Yang, Guandao et al. [17].

To achieve this objective, we will begin by introducing fundamental geometry concepts and their application to Signed Distance Functions. We will then delve into the workings of neural networks, specifically from the perspective of using them as function approximators. We will cover these topics comprehensively from the ground up. Finally, we will connect all of these concepts to explain how neural networks can be used to represent Signed Distance Functions of 3D shapes, and how shape smoothing and sharpening can be performed on them.

Our primary goal is to provide a strong foundation for further exploration of this topic, with a focus on providing a mathematical explanation of these techniques. We believe that this understanding is essential for the continued development of the field.

Chapter 2

Differential geometry of surfaces from distance functions

2.1 Differential Geometry

2.1.1 Surfaces

Before approaching surfaces represented by *Signed Distance Functions* we have to introduce some fundamental concepts of *Differential Geometry*. To do so, we must start by formalizing the concept of a surface in $\mathbb{R}^3$.

Definition 1. ***Parameterized Regular Surface:*** *Let $\Omega \in \mathbb{R}^2$, and the function $X : \Omega \in \mathbb{R}^2 \to \mathbb{R}^3$. We say X is a parameterized regular surface if $(dX)_p$ is injective $\forall p \in \Omega$. Where $(dX)_p$ is the differential of the function X at point p.*

This definition has two drawbacks:

1. We define X as a function, not as a set of points in $\mathbb{R}^3$.
2. It allows self-intersections because bijectivity is not required for the definition.

To address those we introduce the concept of regular surface.

Definition 2. ***Regular Surface:*** *We say a non-empty set, $S \subset \mathbb{R}^3$, is a regular surface if $\forall p \in \mathbb{S}$ it exist an open set, $\mathbb{U} \subset \mathbb{R}^2$, a neighborhood of p, $\mathbb{V} \subset S$, and a differentiable function, $X : \mathbb{U} \to \mathbb{R}^3$, such that:*

(i) $X(\mathbb{U}) = \mathbb{V}$

(ii) $X : \mathbb{U} \to \mathbb{V}$ *is an homeomorphism.*

(iii) $(dX)_q : \mathbb{R}^2 \to \mathbb{R}^3$ *is injective* $\forall q \in \mathbb{U}$

The above definitions lead to the following consequences:

1. $(dX)_q$ is injective $\iff X_u(q) = \dfrac{dX}{du}(q)$ and $X_v(q) = \dfrac{dX}{dv}(q)$ are Linearly Independent (LI). This is equivalent to saying that $||X_u(q) \times X_v(q)|| \neq 0$, which is also equivalent to the existence of a tangent plane (defined in the next section) to the surface at point q.

2. Because of (ii) we need X to be bijective. Hence, exists an inverse function X^{-1}. Moreover, this function will be continuous because X is a homeomorphism.

3. We call the functions X that appear in the definition of regular surface 2, **parametrizations** of the surface. The set of all the parametrizations needed to cover the whole surface $\mathbb{S}$ is called Atlas.

4. Every parametrization of a regular surface is a parameterized regular surface.

2.1.2 Tangent Plane and Surface Normal

In the following section, we will define important concepts to understand how we can use differential geometry to describe a regular surface, S.

Definition 3. ***Tangent Plane*** *Let $S \subset \mathbb{R}^3$ a regular surface and $p \in S$. A vector $\boldsymbol{w} \in \mathbb{R}^3$ is a vector tangent to the surface S iff it exist a paremetrized differentiable curve $\alpha : (-\epsilon, \epsilon) \to S$ such that $\alpha(0) = p$ and $\alpha'(0) = \boldsymbol{w}$. The set of all possible vectors tangent to S at $p \in S$ is denoted as*

$$T_pS = p + \{\boldsymbol{w} \in \mathbb{R}^3 : \text{ it exist } \alpha(t) \subset S \text{ such that } \alpha(0) = p \text{ and } \alpha'(0) = \boldsymbol{w}\}$$

We call T_pS ***tangent plane*** *to S at p. We will often identify it with its director subspace*

$$T_pS \equiv W = \{\boldsymbol{w} \in \mathbb{R}^3 : \text{ it exist } \alpha(t) \subset S \text{ such that } \alpha(0) = p \text{ and } \alpha'(0) = \boldsymbol{w}\}$$

The tangent plane is a linear variety with two dimensions (affine plane) in $\mathbb{R}^3$. We will see it in the following proposition.

Proposition 1. *Let $X : \mathbb{U} \to X(\mathbb{U})$ a parametrization of a regular surface S. Then if $p \in X(\mathbb{U})$ and $q = (u_0, v_0) \in \mathbb{U}$ such that $X(q) = X(u_0, v_0) = p$ then we have that*

$$T_pS = p + (dX)_q(\mathbb{R}^2)$$

That is

$$(dX)_q(\mathbb{R}^2) = \{\boldsymbol{w} \in \mathbb{R}^3 : \text{ it exist } \alpha(t) \subset S \text{ such that } \alpha(0) = p \text{ and } \alpha'(0) = \boldsymbol{w}\}$$

Note that T_pS is then a linear variety that includes p and has as a director subspace $W = (dX)_q(\mathbb{R}^2)$

Before the next proposition we will need a definition.

Definition 4. ***Coordinate curves*** *Let $X : \mathbb{U} \to X(\mathbb{U})$ a parametrization of a regular surface S. Let $p \in X(\mathbb{U})$, hence it exist $q = (u_0, v_0) \in \mathbb{U}$ such that $X(q) = p$. We define the coordinate curves of S in p with respect to X as*

$$\gamma_1(t) = X(u_0 + t, v_0) \qquad \gamma_2(t) = X(u_0, v_0 + t) \tag{2.1}$$

Proposition 2. *Let $X : \mathbb{U} \to X(\mathbb{U})$ a parametrization of a regular surface S. Then if $p \in X(\mathbb{U})$ and $q = (u_0, v_0) \in \mathbb{U}$ such that $X(q) = X(u_0, v_0) = p$ then we have that*

$$T_pS = p + (dX)_q(\mathbb{R}^2) = p + < \{X_u(u_0, v_0), X_v(u_0, v_0)\} >$$

Where $< \{X_u(u_0, v_0), X_v(u_0, v_0)\} >$ is the subspace generated by $\{X_u(u_0, v_0), X_v(u_0, v_0)\}$, that are the partial derivatives with respect to each component from the application X.

Proof. To construct $(dX)_q(\mathbb{R}^2)$, because $(dX)_q$ is linear, we only need to know how it acts over a base of $\mathbb{R}^2$. We take for example $(dX)_q(1, 0)$ and $(dX)_q(0, 1)$. Then $\forall \mathbf{v} = (v_1, v_2) \in \mathbb{R}^2$

$$(dX)_q(\mathbf{v}) = (dX)_q(v_1, v_2) = v_1\,(dX)_q(1, 0) + v_2\,(dX)_q(0, 1)$$

Identifying T_pS with its tangent plane

$$T_pS = (dX)_q(\mathbb{R}^2) =< \{(dX)_q(1, 0), (dX)_q(0, 1)\} >$$

Using the definition of **coordinate curves** 4

$$\gamma_1(t) = X(u_0 + t, v_0) = X((u_0, v_0) + t(1, 0)) \to \gamma_1'(0) = (dX)_{(u_0,v_0)}(1, 0) = (dX)_q(1, 0)$$

$$\gamma_2(t) = X(u_0, v_0 + t) = X((u_0, v_0) + t(0, 1)) \to \gamma_2'(0) = (dX)_{(u_0,v_0)}(0, 1) = (dX)_q(0, 1)$$

So we can write

$$T_pS =< \{(dX)_q(1, 0), (dX)_q(0, 1)\} >=< \{\gamma_1'(0), \gamma_2'(0)\} >$$

To conclude, we have that

$$\gamma_1'(0) = X_u(u_0, v_0) \qquad\qquad \gamma_2'(0) = X_v(u_0, v_0) \tag{2.2}$$

because the coordinate curves just vary in one coordinate, first and second respectively.

This led us to the result

$$T_pS =< \{X_u(u_0, v_0), X_v(u_0, v_0)\} >\equiv p+ < \{X_u(u_0, v_0), X_v(u_0, v_0)\} >$$

□

Definition 5. ***Unit Surface Normal*** *Let $X : \mathbb{U} \subset \mathbb{R}^2 \to S$ a parametrization of a regular surface S. We denote as Unit Normal with respect to X the scalar field $N^X : X(U) \to \mathbb{R}^3$, given by:*

$$N^X(p) = \frac{X_u(q) \times X_v(q)}{||X_u(q) \times X_v(q)||}$$

Where $p \in X(\mathbb{U})$ and $q = X^{-1}(p)$

Remark 1. *The function N^X as defined above has the following properties:*

(i) $||X_u(q) \times X_v(q)|| \neq 0$, *because X is a parametrization of the regular surface, therefore it is injective.*

(ii) $||N^X(p)|| = 1$, *note that it $N^X(p)$ is a vector divided by its modulus.*

(iii) $N^X(p) \in < X_u(q) \times X_v(q) >=< X_u(q), X_v(q) >^{\perp}= T_pS^{\perp} \to N^X(p) \in T_pS^{\perp}$

As we can see, the Unit Surface Normal at point p is orthogonal to every vector in the tangent plane at that point. So it is the normal vector of the tangent plane. Although, by defining the Unit Surface Normal this way, it depends on the parametrization X we choose. This is what denotes the superscript X in N^X. The natural next question is to ask if it really depends on the parametrization we are using.

Proposition 3. *The function Unit Surface Normal, N^X, is not independent of the parametrization, X, chosen to define it. In particular, the magnitude and the direction are indeed independent of X, but the sense is not.*

Proof. To prove the proposition we will answer the following question:

Let X, Y two parametrizations such that $X : U_X \to S$, $Y : U_Y \to S$, with $X(U_X) \cap Y(U_Y) \neq \emptyset$. Take $p \in X(U_X) \cap Y(U_Y)$ then, can we say that $N^X(p) = N^Y(p)$?

The characteristics that define a vector are magnitude, direction, and sense.

- The **magnitude** is the norm of the vector, which is 1, as we have seen in (ii) of remark 1.
- The **direction** is the same because $N^X(p), N^Y(p) \in T_pS^\perp$, as we have seen in (iii) of remark 1, which has only one degree of freedom.

Now we will prove that the **sense** can be different by showing an example.

Let S be a regular surface, $X : U \to S$ a parametrization of S, $p \in S$. Then we have that

$$N^X(p) = \frac{X_u(q) \times X_v(q)}{||X_u(q) \times X_v(q)||}$$

Consider the diffeomorphism $I : \mathbb{R}^2 \to \mathbb{R}^2$ such that $I(x, y) = (y, x)$. Note that $I^2 = I \circ I = id$, where id is the identity function. Then we have that $X \circ I^2 = X \circ id = X$ is a parametrization.

Now we call $Y = X \circ I$ and define it to act on the subspace $I(U) \subset \mathbb{R}^2$, $Y : I(U) \to S$. Note that $I(U)$ is a valid subspace of $\mathbb{R}^2$ because I is a diffeomorphism. Then we have that $X \circ I^2$ acting on U, which is a parametrization of S, is the same that Y acting on $I(U)$. Therefore Y is also a parametrization. Note that $Y(u, v) = X \circ I(u, v) = X(v, u)$

Let $p = Y(t_0, w_0) = X(w_0, t_0) \in S$. Now we have:

$$N^Y(p) = \frac{Y_u(t_0, w_0) \times Y_v(t_0, w_0)}{||Y_u(t_0, w_0) \times Y_v(t_0, w_0)||}$$

Where $Y_u(t_0, w_0) = f'(0)$, with $f(t) = Y(t_0 + t, w_0)$. But, $f'(0) = (Y(t_0 + t, w_0))' = (X(w_0, t_0 + t))' = X_v(w_0, t_0)$. Then $Y_u(t_0, w_0) = X_v(w_0, t_0)$

Analogously, $Y_v(t_0, w_0) = X_u(w_0, t_0)$. Then we have the following:

$$N^Y(p) = \frac{Y_u(t_0, w_0) \times Y_v(t_0, w_0)}{||Y_u(t_0, w_0) \times Y_v(t_0, w_0)||} = \frac{X_v(w_0, t_0) \times X_u(w_0, t_0)}{||X_v(w_0, t_0) \times X_u(w_0, t_0)||} = -N^X(p)$$

So in conclusion, the direction and magnitude are independent of the parametrization, but the sense is not. □

Example 1. ***Unitary sphere*** *Let $S^2(1) = \{(x, y, z) \in \mathbb{R}^3 : x^2 + y^2 + z^2 = 1\}$ be the unitary sphere. We want to calculate the tangent plane to $S^2(1)$ at point $p \in S^2(1)$.*

We have to consider a parametrization that contains p. Because of that, we take e_1, e_2 such that e_1, e_2, p is an orthonormal basis of $\mathbb{R}^3$. Then

$$\begin{aligned} X : D(1) &\longrightarrow S^2(1) \\ (u, v) &\longmapsto X(u, v) = u\, e_1 + v\, e_2 + \sqrt{1 - u^2 - v^2}\, p \end{aligned}$$

where $D1 = (x, y) \in \mathbb{R}^2 : x^2 + y^2 = 1$ is the unitary circle.

X as defined above is a parametrization of $S^2(1)$ with $p = X(0, 0)$.

$$\left.\begin{aligned} X_u(u, v) = e_1 - \frac{u}{\sqrt{1 - u^2 - v^2}}\, p \rightarrow X_u(0, 0) = e_1 \\ X_v(u, v) = e_2 - \frac{v}{\sqrt{1 - u^2 - v^2}}\, p \rightarrow X_v(0, 0) = e_2 \end{aligned}\right\} T_p S^2(1) =< e_1, e_2 >= p^{\perp}$$

Then the tangent plane will be, $\pi_p S^2(1) = p + p^{\perp}$

Definition 6. ***Compatible Parametrizations*** *We know that the direction and magnitude of the normal vector at a point $p \in S$ are independent of the parametrization, but the sense is not.*

Let $X : \mathbb{U} \to S$ and $Y : \mathbb{V} \to S$ two parametrizations of the regular surface S, such that $X(\mathbb{U}) \cap Y(\mathbb{V}) \neq$.

We say that X and Y are compatible if $N^X(p) = N^Y(p)\ \forall p \in X(\mathbb{U}) \cap Y(\mathbb{V})$

Note that in this equality we are only interested in the sense of this vector as we already know that the direction and magnitude will be the same.

Definition 7. ***Orientable surfaces*** *We say that a regular surface S is orientable if we can find an Atlas (set of parametrizations needed to cover the whole surface S) formed by compatible parametrizations.*

2.1.3 Curvature

One of the most important properties of a surface is its curvature, which intuitively tells us how different the surface is from a plane. But it is not trivial how to get this information from a given surface S. Nevertheless, we can use the tools presented in the previous section to measure how

the normal vector of the surface changes. To do so we will consider S to be a regular surface, orientable, and we will choose the Normal Unit Vector to point outwards. In this setting, we do not need the superscript X in N^X, because all the magnitude, direction, and sense (outwards the surface) will be independent of the parametrization. So, from now on we will refer to the application N^X as N.

Definition 8. ***Gauss Map*** *Note that, in reality, the Normal Unit Vector mapping $N : S \to \mathbb{R}^3$, always produces vectors, $N(p)$ of magnitude 1, with an arbitrary direction (depending on the point $p \in S$).*

So they always will be points on the surface of a unitary sphere (radius equal to 1), $S^2(1)$.

So we can rewrite it as $N : S \to S^2(1)$, which is called Gauss Map.

Remark 2. *Given $p \in S$, $dN_p : T_pS \to T_{N(p)}S^2(1)$.*

We saw in the example 1, that $T_{N(p)}S^2(1) = N(p)^\perp$. But $N(p)^\perp = T_pS$.

So the differential map of the Gauss Map at $p \in S$ is an endomorphism, $dN_p : T_pS \to T_pS$

Definition 9. ***Shape Operator***

Let S be a regular surface, orientable and oriented, and $p \in S$. We define the shape operator as the map:

$$W_p := -dN_p : T_pS \to T_pS$$

Definition 10. ***Curvatures***

Let S be a regular surface, orientable and oriented, and $p \in S$. We will use W_p to denote the shape operator as well as the matrix associated with this function interchangeably.

*We denote by **Gaussian curvature** in $p \in S$ to $p \in S$ to $K_g(p) := det(W_p)$.*

*We denote by **mean curvature** in $p \in S$ to $\overline{k}(p) := \frac{tr(W_p)}{2}$.*

*We denote by **principal curvatures** in $p \in S$ to the eigenvalues of W_p, k_1, k_2. The spaces generated by its eigenvectors, $H(k_1), H(k_2)$, are called **principal directions**.*

2.2 Signed Distance Functions

In the following section, we will work with an orientable regular surface in $\mathbb{R}^3$, M. As it is orientable we can choose a direction for its normal vector, N, which will be outward.

Let Ω be a set of points in $\mathbb{R}^3$. We define the distance of any point, $x = (x_1, x_2, x_3) \in \mathbb{R}^3$, to the set Ω as follows:

$$dist(x, \Omega) = \inf_{y \in \Omega} \|x - y\|$$

Using the Euclidean norm for $\mathbb{R}^3$.

We add the constraint of Ω being an open set with non-zero volume and enclosed in a regular surface, $\partial\Omega$. This partition $\mathbb{R}^3$ into three regions, the interior $\Omega^- = \Omega^o = \Omega$, the exterior $\Omega^+ = (\overline{\Omega})^o$, and the boundary $\partial\Omega = \mathbb{R}^3 - \{\Omega^-, \Omega^+\}$. We can then define a distance function for this set as $n(x), \forall x \in \mathbb{R}^3$, given by:

$$n(x) = \begin{cases} dist(x, \partial\Omega), & \text{if } x \in \Omega^+ \\ -dist(x, \partial\Omega), & \text{if } x \in \Omega^- \\ 0, & \text{if } x \in \partial\Omega \end{cases} \tag{2.3}$$

Nevertheless, the above definition does not ensure differentiability $\forall x \in \mathbb{R}^3$, as we can see in the following proof by contradiction.

Proposition 4. ***No differentiable:*** *In the premises stated above, the function $n(x)$ is not differentiable for a point x for which there are two points in $\partial\Omega$ that are at the same minimal distance to x.*

Proof. We assume that $n(x)$ is differentiable in $x \in \mathbb{R}^3$. We will take $x \in \Omega^+$, but the proof is analogous if we take $x \in \Omega^-$.

Consider the case when the minimal distance to $\partial\Omega$ has the same value for two different points $a = (a_1, a_2, a_3)$, $b = (b_1, b_2, b_3) \in \partial\Omega$, mathematically:

$$dist(x, \partial\Omega) = \inf_{y \in \partial\Omega} \|x - y\| = \|x - a\| = \|x - b\|, \text{ with } a \neq b$$

Because $a \neq b$, it must exist at least one component that should be different. Without loss of generality, we assume that this is for example the first component, so $a_1 \neq b_1$. Now, because it is differentiable, we can calculate the derivative of n with respect to the first component of x, which is x_1:

$$\frac{d\,n}{dx_1} = \frac{d}{dx_1}(dist(x, \partial\Omega)) = \frac{d}{dx_1}(\|x - a\|) = \frac{d}{dx_1}(\|x - b\|) \tag{2.4}$$

As we are using the euclidean norm we have that:

$$\frac{d}{dx_1}(\|x-a\|) = \frac{d}{dx_1}(\sqrt{(x_1-a_1)^2+(x_2-a_2)^2+(x_3-a_3)^2}) =$$
$$= \frac{1}{\sqrt{(x_1-a_1)^2+(x_2-a_2)^2+(x_3-a_3)^2}}(x_1-a_1) =$$
$$= \frac{(x_1-a_1)}{\|x-a\|}$$

Doing the analogous with $\frac{d}{dx_1}(\|x-b\|)$ we arrive to:

$$\frac{d}{dx_1}(\|x-a\|) = \frac{(x_1-a_1)}{\|x-a\|} \qquad \frac{d}{dx_1}(\|x-b\|) = \frac{(x_1-b_1)}{\|x-b\|}$$

Using those values in the equation 2.4:

$$\frac{d\,n}{dx_1} = \frac{(x_1-a_1)}{\|x-a\|} = \frac{(x_1-b_1)}{\|x-b\|}$$

But we had that $\|x-a\| = \|x-b\|$, so:

$$x_1 - a_1 = x_1 - b_1 \rightarrow a_1 = b_1$$

which is a contradiction, as we assumed $a_1 = b_1$.

Hence $n(x)$ is not differentiable at the points $x \in \mathbb{R}^3$ for which there are two points in $\partial\Omega$ at the same minimal distance to x.

□

This raises the question of where we can say that $n(x)$, as we defined it in 2.3, is differentiable. The following proposition addresses this question.

Proposition 5. *Given a surface $M = \partial\Omega$ that encloses a domain Ω, the function $n(x)$ is differentiable for all points $x \in \mathbb{R}^3$, except for a set of zero volume.*

To prove proposition 5 we will use the Rademacher's theorem [7] stated below.

Theorem 1. ***Rademacher's theorem:*** *If U is an open subset of $\mathbb{R}^n$ and $f : U \to \mathbb{R}^m$ is Lipschitz continuous, then f is differentiable almost everywhere in U; that is, the points in U at which f is not differentiable form a set of Lebesgue measure zero.*

Note that when $n = 1, 2$ or 3 the Lebesgue measure coincides with the measures of length, area, and volume respectively. As in our case $n = 3$ we can reformulate the above theorem as follows:

Theorem 2. ***Rademacher's theorem for n=3:*** *If U is an open subset of $\mathbb{R}^3$ and $f : U \to \mathbb{R}^m$ is Lipschitz continuous, then f is differentiable almost everywhere in U; that is, the points in U at which f is not differentiable form a set of zero volume.*

Rademacher's theorem enforces Lipschitz continuity, so we define it below.

Definition 11. ***Lipschitz continuity:*** *Let $(X, d_X), (Y, d_Y)$ be two metric spaces. A function, $f : X \to Y$, is K-Lipschitz continuous if it exists a real constant $K > 0$ such that $\forall x_1, x_2 \in X$,*

$$d_Y(f(x_1),\, f(x_2)) \leq K\, d_X(x_1,\, x_2)$$

If a function is Lipschitz continuous it means that it is K-Lipschitz continuous for some $K > 0$.

With that, we can now proceed with the proof of the proposition 5

Proof. Our function is $n : \mathbb{R}^3 \to \mathbb{R}$. Then, if we can prove that it is Lipschitz continuous, we can apply the theorem 2 and the proposition 5 will be proven. So now we prove the Lipschitz continuity of n.

Adapting the definition to our case, we have that the metric spaces are $(\mathbb{R}^3,\, d_e^3)$, and $(\mathbb{R},\, d_e^1)$ with the euclidean distance of each space respectively. Take $K = 1$, $x_1, x_2 \in \mathbb{R}^3$, then we have to prove the following

$$d_e^1(n(x_1),\, n(x_2)) \leq d_e^3(x_1,\, x_2) \to ||n(x_1) - \, n(x_2)||_e^1 \leq ||x_1 - \, x_2||_e^3$$

Let $z \in \partial\Omega$ we have that

$$n(x_1) \leq |n(x_1)| = dist(x_1, \partial\Omega) = \inf_{y \in \partial\Omega} \|x_1 - y\|_e^3 \leq \|x_1 - z\|_e^3$$

Using the triangular inequality of the norm $\|\ \|_e^3$

$$n(x_1) \leq \|x_1 - z\|_e^3 \leq \|x_1 - x_2\|_e^3 + \|x_2 - z\|_e^3 \to n(x_1) - \|x_1 - x_2\|_e^3 \leq \|x_2 - z\|_e^3$$

Because this happens $\forall z \in \partial\Omega$ we can write

$$n(x_1) - \|x_1 - x_2\|_e^3 \leq \inf_{z \in \partial\Omega} \|x_2 - z\|_e^3 = n(x_2) \to n(x_1) - n(x_2) \leq \|x_1 - x_2\|_e^3$$

By doing the same procedure starting with x_2 and $n(x_2)$, we arrive to

$$n(x_2) - n(x_1) \leq \|x_2 - x_1\|_e^3 = \|x_1 - x_2\|_e^3$$

So we can conclude that

$$|n(x_1) - n(x_2)| = ||n(x_1) - n(x_2)||_e^1 =\leq \|x_1 - x_2\|_e^3$$

Therefore n is 1-Lipschitz continuous and the proposition is proven. □

Example 2. *Let $\partial\Omega$ be a ball centered at the origin $O = (0, 0, 0)$ of radius R. Then $\partial\Omega$ is the sphere centered at O. Then, n is not differentiable at O.*

Proof. As O is at the center of the sphere, its distance to any point in the sphere is R.

$$dist(O, x) = R \quad \forall x \in \partial\Omega \to n(O) = -R$$

Take now $p_1 = (1, 0, 0)$, $p_2 = (-1, 0, 0) \in \partial\Omega$.

For the function to be differentiable at O the partial derivatives have to exist and be continuous at that point. Using the definition of partial derivative with respect to the first component we have the following limit:

$$\lim_{\Delta x_1 \to 0} \frac{n((0,0,0)) - n((\Delta x_1, 0, 0))}{\Delta x_1} = \lim_{\Delta x_1 \to 0} \frac{-R - n((\Delta x_1, 0, 0))}{\Delta x_1}$$

which has to be the same regardless of how we approach 0.

We will denote Δx_{1+} and Δx_{1-} when Δx_1 is positive or negative respectively. Then we have that the closest point from the sphere $\partial\Omega$ to $(\Delta x_{1+}, 0, 0)$ will be $(R, 0, 0)$, while the closest to $(\Delta x_{1-}, 0, 0)$ will be $(-R, 0, 0)$. Note that in both cases the function n will be negative because we consider $\Delta x_{1+}, \Delta x_{1-} \to 0$, hence $\Delta x_{1+}, \Delta x_{1-} < R$, so we are inside the surface. This results in the following limits:

$$\lim_{\Delta x_{1+} \to 0} \frac{-R - n((\Delta x_{1+}, 0, 0))}{\Delta x_{1+}} = \lim_{\Delta x_{1+} \to 0} \frac{-R + \sqrt{(\Delta x_{1+} - R)^2}}{\Delta x_{1+}} =$$
$$= \lim_{\Delta x_{1+} \to 0} \frac{-R + |\Delta x_{1+} - R|}{\Delta x_{1+}} = \lim_{\Delta x_{1+} \to 0} \frac{-R + (R - \Delta x_{1+})}{\Delta x_{1+}}$$
$$= \lim_{\Delta x_{1+} \to 0} \frac{-\Delta x_{1+}}{\Delta x_{1+}} = -1$$

$$\lim_{\Delta x_{1-}\to 0} \frac{-R - n((\Delta x_{1-}, 0, 0))}{\Delta x_{1-}} = \lim_{\Delta x_{1-}\to 0} \frac{-R + \sqrt{(\Delta x_{1-} + R)^2}}{\Delta x_{1-}} =$$
$$= \lim_{\Delta x_{1-}\to 0} \frac{-R + |\Delta x_{1-} + R|}{\Delta x_{1-}} = \lim_{\Delta x_{1-}\to 0} \frac{-R + (R + \Delta x_{1-})}{\Delta x_{1-}}$$
$$= \lim_{\Delta x_{1-}\to 0} \frac{\Delta x_{1-}}{\Delta x_{1-}} = 1$$

As we can see the limits are not the same, they depend on the direction we approach O. So the function n is not differentiable at that point.

□

Example 3. *Let $\partial\Omega$ be a torus generated by rotating a circle of radius r whose center is separated by a distance R from the z axis and centered on the origin ($r < R$). Then,*

$$n(x_1, x_2, x_3) = \sqrt{\left(\sqrt{x_1^2 + x_2^2} - R\right)^2 + x_3^2} - r.$$

Proof. First, we have to define mathematically the described torus.

Consider first the plane XY, where $Z = 0$, formed by the points $p = (x, y, 0) \in \mathbb{R}^3$. The distance from these points to the origin will be $\sqrt{x^2 + y^2}$. The distance from p to any point in the circle centered at the origin of radius R in that plane will be $d = |\sqrt{x^2 + y^2} - R|$. As we can see in the figure 2.1 the third component z, d, and r, make a right triangle, therefore they follow the Pythagoras Theorem. So we have that $d^2 + z^2 = r^2$. Hence, we can define the Torus as follows.

$$\partial\Omega = \mathbb{T}^2 = \{(x, y, z) \in \mathbb{R}^3 : (\sqrt{x^2 + y^2} - R)^2 + z^2 = r^2\} =$$
$$= \{(x, y, z) \in \mathbb{R}^3 : \sqrt{(\sqrt{x^2 + y^2} - R)^2 + z^2} - r = 0\}$$

This leads us to a very convenient definition of the torus as an implicit surface that we can use to build the SDF we defined in 2.3. In fact, given a point $x = (x_1, x_2, x_3) \in \mathbb{R}^3$ the expression $\sqrt{(\sqrt{x_1^2 + x_2^2} - R)^2 + x_3^2} - r$ will be 0 when we are on the surface $\partial\Omega$.

Now we consider a point that is inside the torus, $x \in \Omega^-$. Looking at figure 2.1 again, we can see that in this case $d^2 + z^2 < r^2 \equiv \sqrt{(d^2 + z^2} - r) < 0$. Remember that $d = |\sqrt{x_1^2 + x_2^2} - R|$. So the expression $\sqrt{(\sqrt{x_1^2 + x_2^2} - R)^2 + x_3^2} - r$ will be negative. Moreover, its absolute value will give us the distance to $\partial\Omega$.

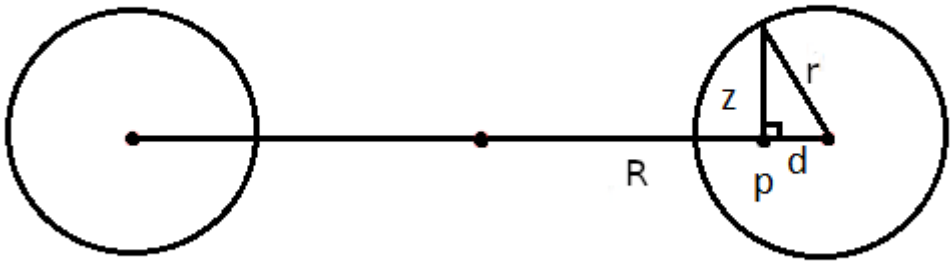

Figure 2.1: Cut in the plane YZ, showing the section of a torus. Adapted from [15]

Lately, we consider a point that is outside the torus, $x \in \Omega^+$. Looking at figure 2.1 again, we can see that in this case $d^2 + z^2 > r^2 \equiv \sqrt{(d^2 + z^2} - r) > 0$. So the expression $\sqrt{(\sqrt{x_1^2 + x_2^2} - R)^2 + x_3^2} - r$ will be positive and will give us the distance to $\partial\Omega$.

Because of that, we can say that the signed distance function as defined in 2.3 is:

$$n(x_1, x_2, x_3) = \sqrt{\left(\sqrt{x_1^2 + x_2^2} - R\right)^2 + x_3^2} - r.$$

Now we will discuss in which points it is not differentiable. We have already proven that the function n as defined in 2.3 is not differentiable in the points where there is more than one point of the surface $\partial\Omega$ at a minimal distance to that point. In the case of a torus, it happens in two cases:

1. At $x = (0, 0, 0)$, which minimal distance to $\partial\Omega$ is $R - r$ (the distance to the smaller circle of the torus intersecting with the plane XY)

2. At $x = (0, 0, x_3)$, that is, all the points in the Z axis. Their distance to the surface is $|\sqrt{(\sqrt{0 + 0} - R)^2 + x_3^2} - r| = |\sqrt{R^2 + x_3^2} - r|$.

3. At all the points of the circle of radius R centered at the origin. Those are inside the torus, and the minimal distance to the surface is r because they are in the center of the circle that rotates to form the torus.

Note that we only make a difference between the first and the second case for conceptual reasons. Strictly speaking, the first case is a particularization of the second one when $x_3 = 0$.

We could use the definition of the derivative as in the previous example, but there is an easier way to do it once we have the function n as we do now. We will use that if a multivariate function is differentiable at a given point, then their partial derivatives with respect to each component exist at that point. So we derive the function $n(x, y, z)$ with respect to the first component x:

$$\frac{d}{dx}(\sqrt{(\sqrt{x^2+y^2}-R)^2+z^2}-r) = \frac{d}{dx}(\sqrt{(\sqrt{x^2+y^2}-R)^2+z^2}) =$$
$$= \frac{\frac{d}{dx}[(\sqrt{x^2+y^2}-R)^2]}{2\sqrt{(\sqrt{x^2+y^2}-R)^2+z^2}} = \frac{2(\sqrt{x^2+y^2}-R)\frac{d}{dx}(\sqrt{x^2+y^2}-R)}{2\sqrt{(\sqrt{x^2+y^2}-R)^2+z^2}} =$$
$$= \frac{(\sqrt{x^2+y^2}-R)\frac{x}{\sqrt{x^2+y^2}}}{\sqrt{(\sqrt{x^2+y^2}-R)^2+z^2}} = \frac{x \cdot \left(\sqrt{x^2+y^2}-R\right)}{\sqrt{x^2+y^2}\sqrt{\left(\sqrt{x^2+y^2}-R\right)^2+z^2}}$$

As we can see in the first and second cases when $x = (0, 0, 0)$ or $x = (0, 0, x_3)$ respectively, we have that $\sqrt{x^2+y^2} = 0$, therefore it gives us a 0 in the denominator. Consequently, the partial derivative does not exist at that point, so the function n is not differentiable.

In the third case, we have that $z = 0$ and $\sqrt{x^2+y^2} = R$, so $\sqrt{\left(\sqrt{x^2+y^2}-R\right)^2+z^2} = 0$, giving us a 0 in the denominator as well. Consequently, the partial derivative does not exist at that point, so the function n is not differentiable. □

Remark 3. *As we have seen the function n as defined in 2.3 does not ensures differentiability for every point $x \in \mathbb{R}^3$. But in virtue of proposition 5 we know that it is not differentiable only in a set of zero volume. Therefore, we can define n in a small $\mathbb{R}^3$-neighborhood, G of M, such that n is differentiable everywhere in G. That is, we take a small $\mathbb{R}^3$-neighborhood, G of M, such that G does not intersect with any non-zero volume set where n is not differentiable.*

This leads us to define G to be small enough so that every point of G lies on some normal ray passing through a point on M, and so that no two normal rays passing through different points on M intersect in G. With that, we ensure that it does not exist two points at the same minimal distance from M. Now we can construct the signed distance function (SDF) for M on G as follows:

$$n(x) = \begin{cases} dist(x, M), & \text{if } x \text{ lies on an outward normal ray of } M \\ -dist(x, M), & \text{if } x \text{ lies on an inward normal ray of } M \\ 0, & \text{if } x \in M \end{cases} \tag{2.5}$$

Definition 12. ***Isosurfaces*** *Let $n(x)$ be a signed distance function of a surface M, as derived above. Then we define the* ***isosurfaces*** *of such function as the level sets:*

$$M_c = \{x \in \mathbb{R}^3 : n(x) = c\}$$

In particular, the zero-isosurface will be the surface M itself:

$$M_0 = M$$

We know that M is a regular surface, then the zero-isosurface, M_0, is also a regular surface. In the next proposition, we will show that this is true for all the isosurfaces, M_c.

Proposition 6. *Let $n(x)$ be a signed distance function of a surface M, as derived in 2.5. Let $c \in \mathbb{R}$ sufficiently small. Let M_c be the c-isosurface, such that*

$$M_c = \{x \in \mathbb{R}^3 : n(x) = c\}$$

and $det(I + c\,dN) \neq 0$ Then the set M_c is a regular surface.

Remark 4. *Note that c has to be small enough so that the c-isosurface is included in the neighborhood G from the definition 2.5.*

If $det(I + c\,dN) = 0$ we can take a smaller c such that $det(I + c\,dN) \neq 0$

Proof. Because M is a regular surface, according to definition 2 we know that $\forall p \in M$ it exist an open set, $\mathbb{U} \subset \mathbb{R}^2$, a neighborhood of p, $\mathbb{V} \subset M$, and a differentiable function, $X : \mathbb{U} \to \mathbb{R}^3$, such that:

(i) $X(\mathbb{U}) = \mathbb{V}$

(ii) $X : \mathbb{U} \to \mathbb{V}$ is an homeomorphism.

(iii) $(dX)_q : \mathbb{R}^2 \to \mathbb{R}^3$ is injective $\forall q \in \mathbb{U}$

For a given $x \in M_c$ we have to find an open set, $\mathbb{U}_x \subset \mathbb{R}^2$, a neighborhood of x, $\mathbb{V}_x \subset \mathbb{M}$, and a differentiable function, $Y : \mathbb{U}_x \to \mathbb{R}^3$, such that it fulfill the three conditions listed before.

Now we fix $x_0 \in M_c$, by how we defined the neighborhood G in 2.5, we know that it exists only one point $p_0 \in M$ such that $dist(x_0, p_0) = c$ and this distance is the minimal distance to M. Moreover, x_0 lies in the normal ray to the surface M that passes through p_0.

Calling $N(p)$ to the surface normal of M in $p \in M$. We can see that $x_0 = p_0 + c\,N(p_0)$. Remember that, because $p_0 \in M$ and M is a regular surface it exists the parametrization X, and the subspaces with the properties described above.

With that, we define,

$$\mathbb{U}_x := \mathbb{U}$$
$$\mathbb{V}_x := \{v + c\,N(v) \,:\, v \in \mathbb{V} \subset M\}$$

$$Y : \mathbb{U}_x \to \mathbb{R}^3$$
$$q \mapsto Y(q) = X(q) + c\,N(X(q))$$

Now we only have to prove that Y satisfies the three requirements:

i) $Y(\mathbb{U}_x) = \mathbb{V}_x$

$$Y(\mathbb{U}_x) = \{Y(q) \,:\, q \in \mathbb{U}_x \subset \mathbb{R}^2\} = \{X(q) + c\,N(X(q)) \,:\, q \in \mathbb{U} \subset \mathbb{R}^2\} =$$
$$= \{v + c\,N(v) \,:\, v \in \mathbb{V} \subset M\} = \mathbb{V}_x$$

ii) $Y : \mathbb{U}_x \to Y(\mathbb{U}_x) = \mathbb{V}$ is an homeomorphism.

To prove this we have to show that Y is continuous and bijective, and its inverse, Y^{-1} is also continuous.

First, we show that Y is continuous. Let q

$$Y(q) = X(q) + c\,N(X(q)) = X(q) + c\,\frac{X_u(q) \times X_v(q)}{||X_u(q) \times X_v(q)||}$$

Because X is a parametrization of the regular surface M, we have that $||X_u(q) \times X_v(q)|| \neq 0$, and that the function $N(p)$ is continuous. So Y is a composition of a polynomial function $(x + c\,y)$ with continuous functions, so it is continuous.

Now we will see that Y is bijective. We know that is surjective because of *i)*, so we only have to prove that is injective. Take $q_1, q_2 \in \mathbb{U}_x$. If $Y(q_1) = Y(q_2)$ then $X(q_1) + c\,N(X(q_1)) =$

$X(q_2) + c\,N(X(q_2))$. On the other hand, because of how we defined the neighborhood G that contains M_c, it exists only one point $p \in M$ such that $dist(Y(q_1), p) = c = dist(Y(q_2), p)$ and this distance is the minimal distance to M. Moreover, $Y(q_1) = Y(q_2)$ lies in the normal ray to the surface M that passes through p. Therefore, $Y(q_1) = X(q_1) + c\,N(X(q_1)) = p + c\,N(p) = X(q_2) + c\,N(X(q_2)) = Y(q_2)$. This means that $p = X(q_1) = X(q_2)$, and because X is bijective we have that $q_1 = q_2$. So Y is injective and surjective, so it is bijective.

Lastly, we will prove that its inverse is continuous. To do this, first, we have to define Y^{-1}. Let $x \in \mathbb{V}_x \subset M_c$, and $p = X(q) \in M$ the only point that in M which distance to x is c, hence $Y(X(q)) = x$. So we only have to define $Y^{-1}(x)$ to be $X^{-1}(p) = q \in \mathbb{U} = \mathbb{U}_x$. We can do it by doing:

$$\begin{aligned} Y^{-1} : \mathbb{V}_x &\to \mathbb{U}_x \\ x &\mapsto Y^{-1}(x) = X^{-1}(x - c\,N(X(q))) \end{aligned}$$

See that it is well defined as we can always find a single $p = X(q) \in M$ and therefore, a single $p = X(q) \in \mathbb{U}$ associated with every $x \in \mathbb{V}_x$.

We can see that we accomplish the desired function by doing:

$$Y^{-1}(Y(q)) = Y^{-1}(X(q) + c\,N(X(q))) = X^{-1}(X(q) + c\,N(X(q)) - c\,N(X(q))) = \tag{2.6}$$

$$= X^{-1}(X(q)) = q = Id(q) \tag{2.7}$$

As X^{-1} is continuous, because X is an homeomorphism, Y^{-1} is a composition of continuous functions and therefore continuous.

iii) $(dY)_q : \mathbb{R}^2 \to \mathbb{R}^3$ is injective $\forall q \in \mathbb{U}_x$

Because the differential is linear we can write:

$$dY = d(X + c\,N(X)) = dX + c\,dNdX = (I + c\,dN)dX$$

By identifying the differential with its associated matrix, to see that $(dY)_q$ is injective is enough to ensure that $det((dY)_q) \neq 0$. Calculating $det((dY)_q)$,

$$det((dY)_q) = det((I + c\,dN)dX) = det(I + c\,dN)\,det(dX)$$

We know that $det(dX) \neq 0$ because X is a parametrization of the regular surface and $det(I + c\,dN) \neq 0$ by hypothesis. Hence $det((dY)_q) \neq 0$, so $(dY)_q$ is injective.

□

Remark 5. *In the previous proof, we restricted c such that* $det(I + c\,dN) \neq 0$. *This is needed to ensure that* $det((dY)_q) \neq 0$ *and therefore the surface* M_c *is regular. But we can say even more, there will be some value of c, we call it* c_0, *such that if* $|c| < |c_0|$ *then the surface* M_c *is regular.*

Proof. We study $det(I + c\,dN)$, but note that those are 2×2 matrices and that $dN = -W$, where W is the Weingarten matrix. In the appropriate base, we can write $W = \begin{pmatrix} k_1 & 0 \\ 0 & k_2 \end{pmatrix}$

$$det(I + c\,dN) = det(I - c\,W) = det\left(\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - c\begin{pmatrix} k_1 & 0 \\ 0 & k_2 \end{pmatrix}\right) =$$

$$= \begin{vmatrix} 1 - ck_1 & 0 \\ 0 & 1 - ck_2 \end{vmatrix} = (1 - ck_1)\,(1 - ck_2) = 1 - ck_2 - ck_1 + c^2k_1k_2 = c^2k_1k_2 - c(k_1 + k_2) + 1$$

Taking c as a variable we can now see for which values of c the expression is 0.

$$c = \frac{(k_1 + k_2) \pm \sqrt{(k_1 + k_2)^2 - 4(k_1k_2)1}}{2k_1k_2} = \frac{(k_1 + k_2) \pm \sqrt{k_1^2 + k_2^2 + 2k_1k_2 - 4k_1k_2}}{2k_1k_2} =$$

$$= \frac{(k_1 + k_2) \pm \sqrt{k_1^2 + k_2^2 - 2k_1k_2}}{2k_1k_2} = \frac{(k_1 + k_2) \pm \sqrt{(k_1 - k_2)^2}}{2k_1k_2} = \frac{(k_1 + k_2) \pm |k_1 - k_2|}{2k_1k_2} =$$

$$= \begin{cases} \dfrac{k_1 + k_2 + k_1 - k_2}{2k_1k_2} = \dfrac{2k_1}{2k_1k_2} = \dfrac{1}{k_2} \\ \dfrac{k_1 + k_2 - k_1 + k_2}{2k_1k_2} = \dfrac{2k_2}{2k_1k_2} = \dfrac{1}{k_1} \end{cases}$$

Now, for every point $p \in M$ we define $k(p) = max\{k_1(p), k_2(p)\}$, that is the maximum of the two principal curvatures. Then if it exists we can take

$$k = \max_{p \in M}\{k(p)\}$$

If we take $c_0 < \dfrac{1}{k}$, then we have that if $|c| < |c_0|$ then $det(I + c\,dN) \neq 0$, so M_c is a regular surface. □

2.2.1 Surface Normal of Signed Distance Functions

We have already introduced the concept of surface normals for regular surfaces. The next step is applying this to the specific surfaces described by the signed distance functions defined above.

Given a point $x \in G$, $y(x)$ will be the unique point on M whose normal ray passes through x, and $v(x)$ will be the outward unit normal to M at $y(x)$. Note that, because of how we have defined G, there will be only one point of M whose normal ray passes through x, which is $y(x)$, so the function is well defined. Hence, the line of the minimal distance between x and M goes through $y(x)$. Moreover, it is perpendicular to M. So the vector that goes from $y(x)$ to x is the normal unitary vector to M in $y(x)$, which is $v(x)$; with a magnitude equal to the distance from x to $y(x)$, which has been defined as $n(x)$. This yields the following equality:

$$x - y(x) = n(x)\, v(x)$$

In the following, we will denote $y(x) = y, n(x) = n$ and $v(x) = v$. We can derive this vector equation with respect to the first coordinate of the point x, x_1, to obtain:

$$\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} - \frac{dy}{dx_1} = \frac{dn}{dx_1} v + n \frac{dv}{dx_1}$$

We apply the dot product by v on both sides of the equation, taking into account that $v \cdot v = \sqrt{\|v\|} = 1$, as v is a unitary vector.

$$v_1 - \frac{dy}{dx_1} \cdot v = \frac{dn}{dx_1} + n \frac{dv}{dx_1} \cdot v$$

As $y \in M$, any derivative of y must be tangent to M. Because v is, by definition, orthogonal to M we have that:

$$\frac{dy}{dx_1} \cdot v = 0$$

Because $\|v\| = 1$ by deriving both sides with respect to the first component we have:

$$\frac{dv}{dx_1} = 0 \rightarrow \frac{dv}{dx_1} \cdot v = 0$$

Resulting in the following equality:

$$\frac{dn}{dx_1} = v_1$$

Repeating the process, but deriving with respect to x_2 and x_3, we arrive to the following conclusion:

$$\nabla n = \begin{pmatrix} \frac{dn}{dx_1} \\ \frac{dn}{dx_2} \\ \frac{dn}{dx_3} \end{pmatrix} = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} = v$$

Therefore:

$$\|\nabla n\| = \|v\| = 1 \tag{2.8}$$

This is a particular case of the Eikonal equation which is $\|\nabla n\| = f(x)$. We have proven that this should hold true for every signed distance function n. So we can use it to ensure that a field remains a valid SDF throughout any manipulation.

To summarize we have the following situation:

1. ∇n is perpendicular to all level surfaces of n.
2. $M_0 = \{x \in \mathbb{R}^3 : n(x) = 0\} = M \rightarrow$The zero-isosurface of n is the surface M itself.
3. $\|\nabla n\| = 1$
4. The direction of increase of n is outward the surface is outward M, because of how it is defined.

This leads us to conclude that the vector ∇n coincides with the outward unit normal vector at $p \in M$, $N(p)$. We can therefore extend our definition of Unit Surface Normal 5, for this particular case by setting:

$$\boxed{N(p) = \nabla n(p) = \begin{pmatrix} n_1(p) \\ n_2(p) \\ n_3(p) \end{pmatrix}} \tag{2.9}$$

where $n_i(p) = \dfrac{dn}{dx_i}(p)$.

2.2.2 Curvature of Signed Distance Functions

The previous section ended with the equation 2.9. Therefore if we call J_N the Jacobian matrix of N, this will be equivalent to the Hessian matrix of the SDF n, denoted by H_n. Note that both J_N and H_n are 3×3 matrices.

Remember that we defined the *Shape operator* in 9 as follows:

$$W_p := -dN_p : T_pS \to T_pS$$

Remember also that we used W_p to denote both, the function and the matrix associated with the function. So, if we restrict J_N and H_n to the vectors from T_pS, we can say that,

$$W_p = -dN_p \equiv -J_N = -H_n$$

This allows us to formulate the curvatures defined 10 based on these matrices. These definitions are not direct, since W_p is a 2×2 matrix and J_N is 3×3. In the following proposition, we will see how the **mean curvature**, $\overline{k}(p)$ can be defined in terms of $J_N = H_n$.

Proposition 7. *The* ***mean curvature*** *in $p \in S$ is* $\overline{k}(p) := \dfrac{tr(W_p)}{2} = \dfrac{tr(-H_n)}{2} = -\dfrac{tr(H_n)}{2}$.

Proof. To prove this we will prove that $\dfrac{tr(W_p)}{2} = -\dfrac{tr(J_N)}{2}$. Since we know that $tr(J_N) = \sum_{i=1}^{3} \lambda_i$, where λ_i are the eigenvalues of J_N, we need to find those values (we can do it because the matrix is symmetric, hence diagonalizable).

We know that $< N(p), N(p) >= 1$ by differentiating:

$$\frac{d}{dx_i}(< N(p), N(p) >) =< \frac{d}{dx_i}(N), N > + < N, \frac{d}{dx_i}(N) >= 2 < \frac{d}{dx_i}(N), N >= 0$$

So $< \frac{d}{dx_i}(N), N >= 0 \, \forall i = 1, 2, 3$. Knowing that $J_N = \begin{pmatrix} \frac{dN}{dx_1} \\ \frac{dN}{dx_2} \\ \frac{dN}{dx_3} \end{pmatrix}$, which is a 3×3 matrix:

$$J_N N = 0$$

This gives us the first eigenvector (N) and its eigenvalue (0).

To obtain the other two let $\alpha : I \to M$ a curve on the surface M, $\alpha(s) = p$, and $X : U \to M$ the parametrization of the surface that includes $\alpha(s)$. Then $\alpha'(s) \in T_pS$, and if $\alpha'(s)$ is a principal direction we know that it is an eigenvector of W_p, so being k_i the principal curvature associated with that direction we have:

$$W_p \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} = k_i \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} k_i \, a_1 \\ k_i \, a_2 \end{pmatrix} \tag{2.10}$$

Where $\alpha'(s) = a_1 \, X_u + a_2 \, X_v$. We can do this because $\alpha'(s) \in TpS$ and TpS is generated by $\{X_u, X_v\}$.

On the other hand, we know that $W_p \equiv -J_N$, so:

$$W_p \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} \equiv -J_N \alpha'(s) = -J_N \left(a_1 \, X_u + a_2 \, X_v \right)$$

Then because equation 2.10 we have that:

$$-J_N \alpha'(s) = (k_i \, a_1 \, X_u + k_i \, a_2 \, X_v) = k_i \alpha'(s) \to J_N \alpha'(s) = -k_i \alpha'(s)$$

Therefore, $\alpha'(s)$ is an eigenvector of J_N and $-k_i$ is an eigenvalue. This is true for $i = 1, 2$, hence the eigenvalues of J_N are $\{0, k_1, k_2\}$.

With that, we can say that $tr(J_N) = 0 + k_1 + k_2 = tr(W_p)$

We can conclude that $\overline{k}(p) := \dfrac{tr(W_p)}{2} = \dfrac{tr(-J_N)}{2} = -\dfrac{tr(J_N)}{2} = -\dfrac{tr(H_n)}{2}$ $\square$

Chapter 3

Neural Networks and Neural Fields

Neural networks are at the core of Deep Learning advances in the last decade. Inspired by the human brain, they use a combination of artificial neurons to process and analyze data inputs, allowing them to learn from experience and improve their performance over time. In this chapter, we will see what all of this means, as well as a special type of neural network called *neural fields*, which are very convenient for geometric and spatial representations.

3.1 Neural Networks

First, we will cover the fundamental concepts of neural networks, including activation functions, gradient descent, and backpropagation. We will start with a simple example to detail all the steps involved in training. For now, the reader just has to think about a neural network as a function $F(x)$ that takes as input x and produces an output y. This function F is determined by a set of parameters P.

3.1.1 Gradient Descent

To explain how gradient descent works we will use a simple example that has nothing to do with neural networks. Let $F(\omega) = (\omega - 1)^2$ be a function for which we want to find the value for ω that minimizes the value of the function.

By calculating its derivative $F'(\omega) = 2(\omega - 1) \rightarrow \omega* = 1$, because it is the value for which the derivative becomes 0.

For neural networks, the function is more complex, we have more parameters to optimize and we find multiple minimums, so we use gradient descent.

We start with a random value for ω; for example $\omega = 2$ and $F(2) = 1$. Now we have to make a small change to ω in order for $F(\omega)$ to be smaller. So we could increase or decrease ω. The answer to what to do is given by the derivative of the function, which tells us the direction in which we have to move ω for F to increase. Because of that, to make the value of the function smaller we will update ω adding an increment of $\Delta\omega = -f'(\omega)$.

Starting from ω_0, the next step will give us $\omega_1 = \omega_0 - f'(\omega_0)$, and so on, until $\omega_n = \omega_{n-1} - f'(\omega_{n-1})$. Once we stop, we expect the value $F(\omega_n)$ to be as close to the minimum as possible. Nevertheless, to avoid the increment $\Delta\omega$ being too big, we set a value of $\alpha > 0$, such that

$$\omega_i = \omega_{i-1} - \alpha f'(\omega_{n-1}).$$

This α is known as the learning rate and determines how much we move in the direction in which the function decreases. Its value depends on the problem, and we should try different values. The reason is that with more complex functions we can find local minima in which we will get stuck if the learning rate is too close to 0. Although, if we take a learning rate that is too high, the changes in ω will be too big and we will miss the minimum we are trying to reach.

For neural networks, the function we will try to optimize will not be a function of the inputs x or outputs y. Instead, it will be a function of the parameters P of the network. As we have defined P as a set, that means that we can have multiple parameters, in practice we have thousands or more. Hence we will apply gradient descent using the partial derivatives with respect to each one of the parameters in P.

3.1.2 Multilayer Perceptron

Deep feedforward networks, also called feedforward neural networks or multilayer perceptrons (MLPs), are the quintessential deep learning models [5]. They take an input vector $\mathbf{x} = (x_1, x_2, ..., x_n)$, and produce an output vector $\mathbf{y} = (y_1, y_2, ..., y_l)$. The goal of MLPs is to approximate a function $F^*(\mathbf{x}) = \mathbf{y}$.

Finally, to see what an MLP looks like one can see figure 3.2. To understand this better we have to introduce some notation.

This is a five-layer MLP, as we name each column in the diagram as a *layer*. In particular, the first layer is the inputs and the last one is the outputs. All the intermediate layers are called *hidden layers*. We will index each layer with a superscript $a^{(k)}$, with $k = 0, ..., 4$. Note that this way, we will write the input and output layers with the same notation as the hidden

layers, where $x_i = a_i^{(0)}$, with $i = 1, ..., n$; and $y_j = a_j^{(4)}$, with $j = 1, ..., l$. Each layer can have m_k nodes, being $m_0 = n$ and $m_4 = l$.

Each connection between two nodes represents a parameter called *weight*, $w_{i,j}^{(k)} \in P$. This is the parameter of the connection between node i from layer k and node j from layer $k+1$. So here $i = 1, ..., m_k$ and $j = 1, ..., m_{k+1}$. For each node (or neuron) we have an additional parameter, $b_j^k \in P$, with $j = 1, ..., m_{k+1}$. Those are called *biases* and note that we have $k+1$ of them because each of those is associated with one neuron in the layer $k+1$.

To combine those parameters, for each neuron we multiply the weight of each incoming connection by the value of the node that comes from and add the result. Lately, we add the bias of that neuron and we apply a non-linear function σ, normally referred to as **activation function**. This produces the output of the neuron which is also called activation. Then the outputs (activations) of one layer serve as input to the next one, and we continue this process until we reach the final layer. An illustration of the computation of the first layer can be found in 3.3. Here it can also be seen how this operation can be conveniently expressed as a matrix multiplication, which makes them very efficient computationally.

There are multiple options for choosing the activation function σ, we can see some of those in figure 3.1. Choosing one of them will strongly depend on our problem and there is no universal agreement about which one works best. Nevertheless, it can be empirically tested and stated that some of them work better for specific problems, as we will see on *Chapter 4*.

Remark 6. ***On differentiability:*** *In the previous section, we talked about gradient descent, and we saw that relies on derivatives. So, when applying it to MLP one would expect all the operations to be differentiable. But as we can see in 3.1, there are some activation functions that are not differentiable at some points.*

For instance, take the ReLU activation which is mathematically expressed as $ReLU(x) = max\{0, x\}$. *We can see that this is not differentiable at* $x = 0$ *by using the definition of the derivative when we are approaching* 0 *with* $x < 0$ *and* $x > 0$*:*

$$\lim_{x^- \to 0} \frac{max\{0, x^-\} - max\{0, 0\}}{x^- - 0} = \lim_{x^- \to 0} \frac{0}{x^-} = 0$$

$$\lim_{x^+ \to 0} \frac{max\{0, x^+\} - max\{0, 0\}}{x^+ - 0} = \lim_{x^+ \to 0} \frac{x^+}{x^+} = 1$$

So we cannot compute the derivative in $x = 0$ *if we are using this activation function. This may be heuristically justified by observing that gradient-based optimization on a digital computer is subject to numerical error anyway. When a function is asked to evaluate g(0), it is very unlikely that the underlying value truly was 0. Instead, it was likely to be some small value* ϵ *that was rounded to 0 [5]. So it is justified to use this function, and similar explanations can be given in other cases.*

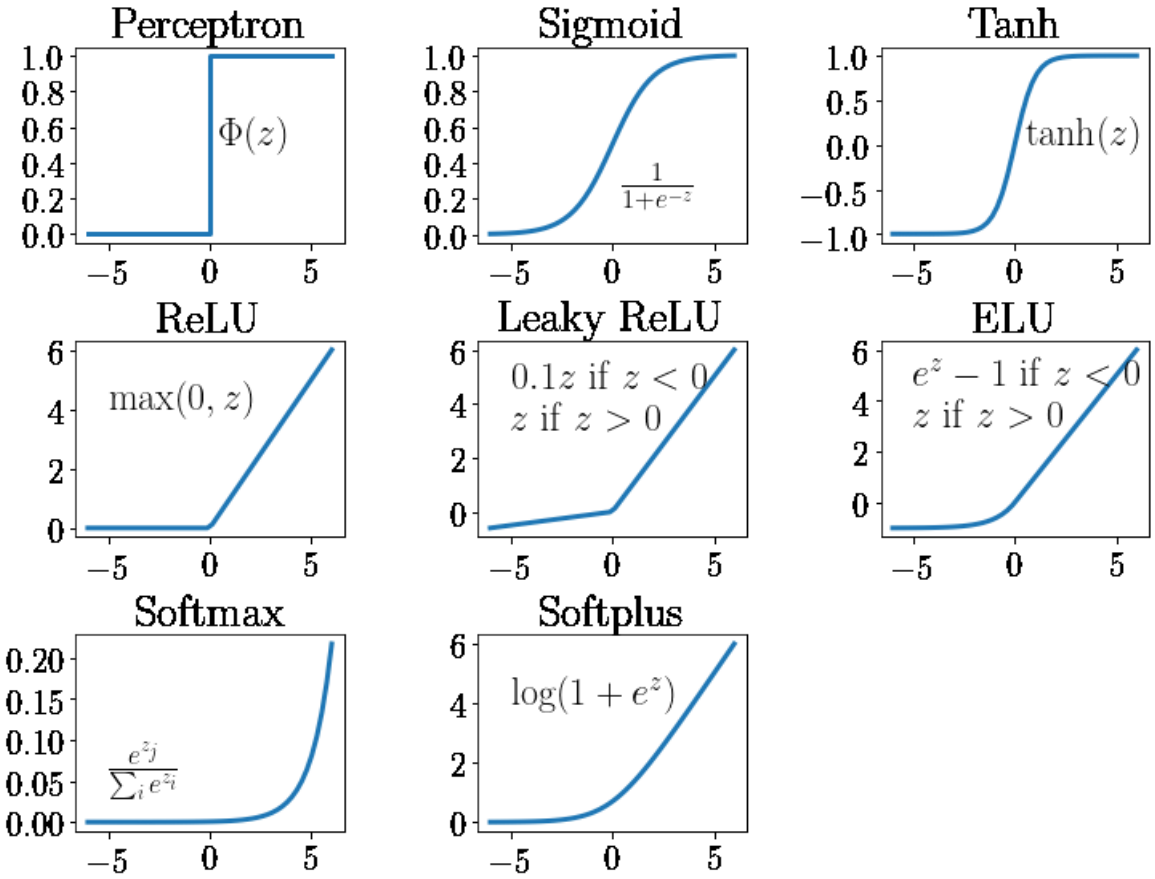

Figure 3.1: Different activation functions to apply in the hidden layers. Image from [9]

It would be valid to ask why we need those activation functions to be applied between the layers. The answer will be given in Remark 8, but first, we need to define two types of functions.

Definition 13. ***Linear function:*** *Let $F : \mathbb{R}^n \to \mathbb{R}^m$ be a function, such that for $\boldsymbol{u} \in \mathbb{R}^n$, $F(\boldsymbol{u}) \in \mathbb{R}^m$. F is a linear function only if:*

$$F(a\boldsymbol{u} + b\boldsymbol{v}) = a\,F(\boldsymbol{u}) + b\,F(\boldsymbol{v}) \qquad \forall \boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^n, \forall a, b \in \mathbb{R}$$

Proposition 8. *Let A be an $m \times n$ matrix, $A \in \mathbb{R}^{m \times n}$. Let $F_A : \mathbb{R}^n \to \mathbb{R}^m$ be a function, such that for $\boldsymbol{u} \in \mathbb{R}^n$, $F_A(\boldsymbol{u}) = A\boldsymbol{u} \in \mathbb{R}^m$. Then the function F_A is a linear function.*

Consequently, we can state that multiplying a vector by a matrix is a linear transformation between $\mathbb{R}^n$ and $\mathbb{R}^m$.

Proof. Let $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$, $a, b \in \mathbb{R}$ then using the matrix properties we have:

$$F_A(a\mathbf{u} + b\mathbf{v}) = A\,(a\mathbf{u} + b\mathbf{v}) = a(A\mathbf{u}) + b(A\mathbf{v}) = a\,F(\mathbf{u}) + b\,F(\mathbf{v})$$

□

Definition 14. ***Affine function:*** *Note that we should define the function F to go from one affine space to another. So we denote with $\mathbb{R}^n$ and $\mathbb{R}^m$ the affine spaces ($\mathbb{R}^n$, $\mathbb{R}^n$, g_e) and ($\mathbb{R}^m$, $\mathbb{R}^m$, g_e) respectively, where g_e is the Euclidean metric. To distinguish when we are using points or vectors, we will denote the vectors with bold letters.*

Let $F : \mathbb{R}^n \to \mathbb{R}^m$ be a function, such that for $u \in \mathbb{R}^n$, $F(u) \in \mathbb{R}^m$. F is an affine only if it exists $o \in \mathbb{R}^n$, treated as a point, and a linear function $\overline{F} : \mathbb{R}^n \to \mathbb{R}^m$ such that:

$$\overline{F}(\overline{ou}) = \overline{F(o)F(u)} \qquad \forall \boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^n$$

Where $\overline{uv} = v - u \,\forall u, v \in \mathbb{R}^n$, which is the vector from $\mathbb{R}^n$ beginning at point u and ending at point v. It is analogous when $u, v \in \mathbb{R}^m$.

The linear function $\overline{F}$ is called associated linear function.

Remark 7. *For any affine function F we have that:*

$$\overline{F}(\overline{ou}) = \overline{F(o)F(u)} = F(u) - F(o) \qquad \forall \boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^n$$

Hence we can write, $F(u) = F(o) + \overline{F}(\overline{ou})$.

Proposition 9. *Let A be an $m \times n$ matrix, $A \in \mathbb{R}^{m\times n}$, and $b \in \mathbb{R}^m$. Let $F : \mathbb{R}^n \to \mathbb{R}^m$ be a function, such that for $u \in \mathbb{R}^n$, $F_A(u) = Au + b \in \mathbb{R}^m$. Then the F is an affine function with associated linear function $\overline{F} : \mathbb{R}^n \to \mathbb{R}^m$, such that $\overline{F}(\boldsymbol{u}) = A\boldsymbol{u}$.*

Consequently, we can state that multiplying a vector by a matrix and adding another vector is an affine transformation between $\mathbb{R}^n$ and $\mathbb{R}^m$.

Proof. Let $o = 0$ be the zero of $\mathbb{R}^n$, $u \in \mathbb{R}^n$, both treated as points, then we have that:

$$\overline{F}(\overline{ou}) = \overline{F}(\mathbf{u}) = A\mathbf{u}$$

$$\overline{F(o)F(u)} = \overline{(Ao + b)(Au + b)} = \overline{b(Au + b)} = (Au + b - b) = A\mathbf{u}$$

So we see that those are equal which finishes the proof. □

Affine transformations between Euclidean spaces can be seen as linear transformations followed by a translation.

In figure 3.3 we can see that before applying σ we have performed an affine transformation of the inputs.

Proposition 10. *Given the affine functions $\varphi : A \to A'$ $\psi : A' \to A''$, the composition $\varphi \circ \psi : A \to A''$ is also an affine function. Its associated linear function is $\overline{\varphi \circ \psi} = \overline{\varphi} \circ \overline{\psi}$.*

Proof. As φ and ψ are affine we can write:

$$\varphi(a_1) = \varphi(o_1) + \overline{\varphi}(\overline{o_1 a_1}) \qquad \psi(a_2) = \psi(o_2) + \overline{\psi}(\overline{o_2 a_2})$$

Let $u \in A$, then we can write:

$$\overline{\varphi \circ \psi(o_1)\varphi \circ \psi(u)} = \overline{(\psi(\varphi(o_1)))(\psi(\varphi(u)))} = \psi(\varphi(u)) - \psi(\varphi(o_1)) =$$
$$= \psi(o_2) + \overline{\psi}(\overline{o_2 \varphi(u)}) - \psi(o_2) - \overline{\psi}(\overline{o_2 \varphi(o_1)}) = \overline{\psi}(\varphi(u) - o_2) - \overline{\psi}(\varphi(o_1) - o_2)$$

As $\overline{\psi}$ and $\overline{\varphi}$ are linear, we can write:

$$\overline{\psi}(\varphi(u) - o_2) - \overline{\psi}(\varphi(o_1) - o_2) = \overline{\psi}(\varphi(u) - (o_2) - \varphi(o_1) + o_2) = \overline{\psi}(\varphi(u) - \varphi(o_1)) =$$
$$= \overline{\psi}(\varphi(o_1) + \overline{\varphi}(\overline{o_1 u}) - \varphi(o_1) - \overline{\varphi}(\overline{o_1 o_1}))) = \overline{\psi}(\overline{\varphi}(\overline{o_1 u}) - \overline{\varphi}(\overline{0})) =$$
$$= \overline{\psi}(\overline{\varphi}(\overline{o_1 u})) = \overline{\varphi} \circ \overline{\psi}(\overline{o_1 u})$$

So taking as *associated linear function* $\overline{\varphi \circ \psi} = \overline{\varphi} \circ \overline{\psi}$ we have that:

$$\overline{\varphi \circ \psi(o_1)\varphi \circ \psi(u)} = \overline{\varphi} \circ \overline{\psi}(\overline{o_1 u}) = \overline{\varphi \circ \psi}(\overline{o_1 u})$$

□

Remark 8. ***The importance of the non-linear activation function:*** *It is essential to apply the non-linear function to the result of the operation of each neuron.*

This is because, as shown in figure 3.3, if we do not apply the function σ the result of each neuron is just an affine transformation. In addition, as we have seen in proposition 10, the composition of affine transformations is also affine. Therefore, if we do not apply σ we will only be able to approximate affine functions. This is not convenient because we want to be able to approximate more complex functions.

In fact, if the function σ is linear (instead of non-linear). We have the same result, as linear transformations are a subset of affine transformations.

An interesting property of such models is the *Universal Approximation Theorem*, proven in [8], and stated below:

Theorem 3. ***Universal Approximation Theorem*** *Multilayer feedforward networks with as few as one hidden layer using arbitrary activation functions are capable of approximating any Borel measurable function from one finite dimensional space to another to any desired degree of accuracy, provided sufficiently many hidden units are available. In this sense, multilayer feedforward networks are a class of universal approximators.*

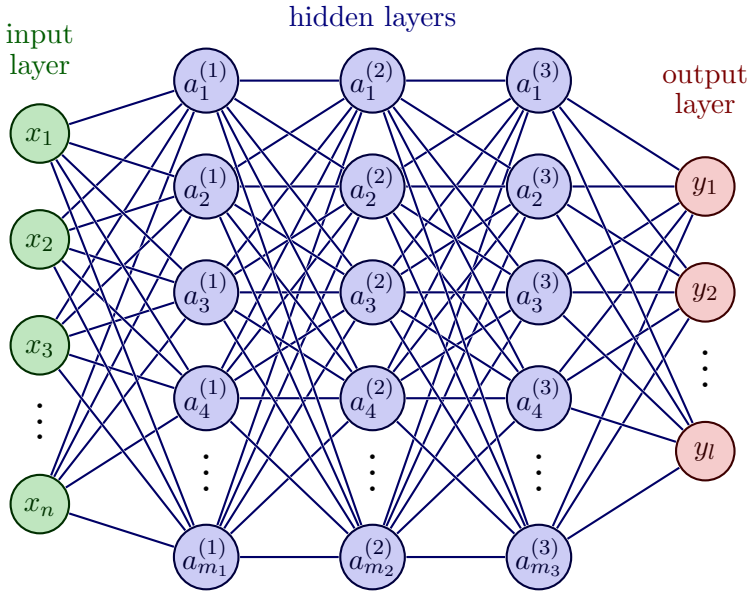

Figure 3.2: General Architecture Multilayer Perceptron with three hidden layers. Figure adapted from [13]

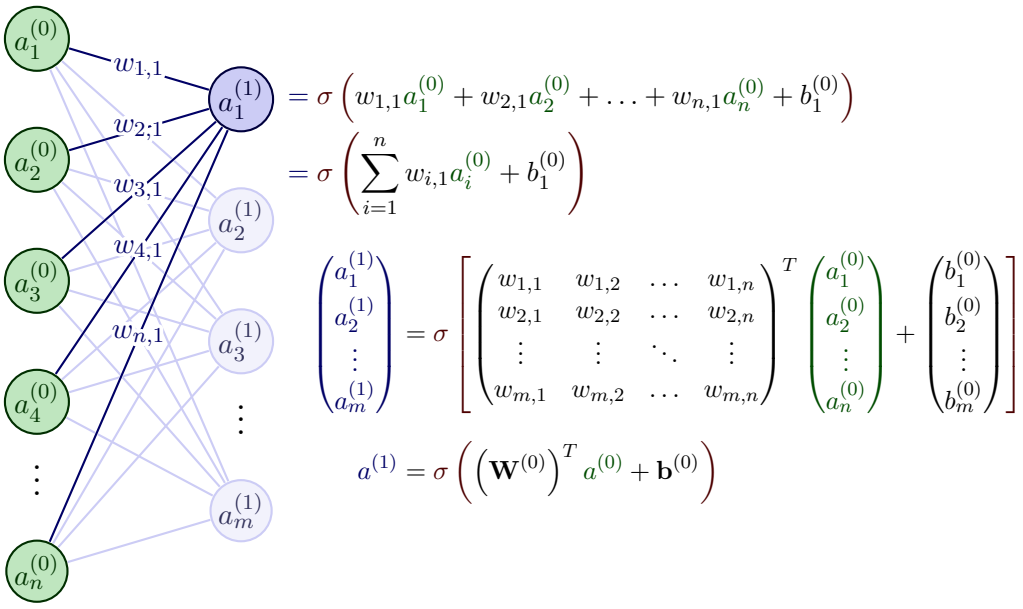

Figure 3.3: Activation computation MLP. This figure shows how the activations of the first layer are computed, as well as its matrix representation. The activation function σ is applied **elementwise** to the components of the vector. Note that we do not use the sub-index in a_m for simplicity, but if there were more layers this would be a_{m_1}. Figure adapted from [13]

.

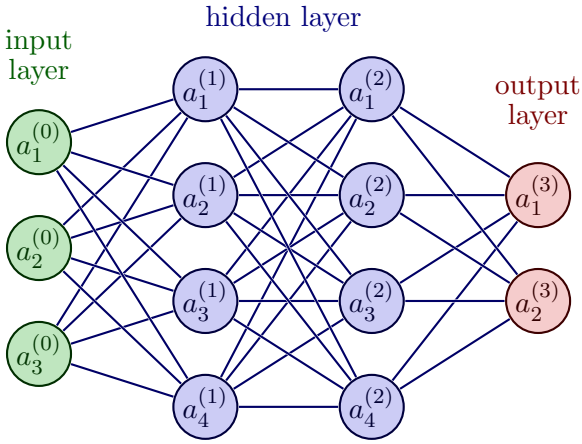

Figure 3.4: Simple Multilayer Perceptron

It will be very useful to think about MLP as universal approximators in the next sections.

3.1.3 Back Propagation

We have already seen in section 3.1.1 how gradient descent works, this is the (most basic) optimization method used in the learning algorithm of the neural networks, but we have not explained which algorithm we use. This is called *Back Propagation Algorithm*. In this section, we will apply it to a simple MLP to illustrate all the steps.

We will work with the MLP shown in figure 3.4. It has four layers, two of which are hidden. For simplicity, we will use the sigmoid activation function, defined as follows:

$$\sigma(x) = (1 + e^{-x})^{-1} = \frac{1}{1 + e^{-x}}$$

Then we can easily calculate the derivative, resulting in:

$$\sigma'(x) = \frac{-e^{-x} \cdot (-1)}{(1 + e^{-x})^2} = (1 + e^{-x})^{-1} \frac{e^{-x}}{1 + e^{-x}} = \sigma(x) \frac{1 - 1 + e^{-x}}{1 + e^{-x}} =$$
$$= \sigma(x) \left(\frac{-1}{1 + e^{-x}} + \frac{1 + e^{-x}}{1 + e^{-x}}\right) = \sigma(x)(1 - \sigma(x))$$

So we have that for computing the derivative of this function we do not need to actually calculate it, but we can use $\sigma'(x) = \sigma(x)(1 - \sigma(x))$.

We will denote the weights and biases from the connections between layer k (previous) and layer $k + 1$ (next) with the superscript (k). For the weights, the subscript will be i, j, where

i is the number of the node from the previous layer, k, and j is the node from the next layer, $k+1$. For the biases, the subscript will be j and will mean the neuron with which is associated from layer $k+1$. For an example see figure 3.3, but note that for all the weights, $w_{i,j}$ they should be $w^{(0)}_{i,1}$, as we are between layer 0 and layer 1. The ranges for those indexes in the case of our simple example from figure 3.4 will be $k = 0, ..., 3$, and the i, j will vary depending on the layers we are connecting. Being n_p, n_n, the number of nodes for the previous and the next layer respectively, $i = 1, ..., n_p$, $j = 1, ..., n_n$. For example, for the connections between layer 0 and layer 1, $i = 1, ..., 4$ and $j = 1, ..., 3$.

Remember that we denoted the inputs, $\mathbf{x} = (x_1, x_2, x_3)$, and outputs, $\mathbf{y} = (y_1, y_2)$, as the activations $a^{(0)}$ and $a^{(3)}$ respectively. This means that $\mathbf{x} = a^{(0)}$ and $\mathbf{y} = a^{(3)}$.

Remark 9. ***Loss***

Before continuing we have to introduce the notion of loss. For a given input $\boldsymbol{x}$, we know that we want to get a desired output, $\boldsymbol{s} = (s_1, s_2)$, which is called target (when dealing with classification problems they are more often called labels). But the output that we get from the neural network is $MLP(x) = \boldsymbol{y}$, which is probably not the same as $\boldsymbol{s}$. Therefore we want to measure how wrong was our prediction, and we make it by using a loss (or error) function $e(\boldsymbol{y}, \boldsymbol{s})$. In our case, we will use a simple function, the Mean Squared Error, which is based on the distance between $\boldsymbol{y}$ and $\boldsymbol{s}$:

$$e(\boldsymbol{y}, \boldsymbol{s}) = \frac{1}{2}(s_1 - y_1)^2 + \frac{1}{2}(s_2 - y_2)^2$$

Intuitively, this gives us how "far" we are from the expected output. Therefore this is the function we want to minimize using gradient descent: $e(MLP(\boldsymbol{x}), \boldsymbol{s})$. And we will minimize it with respect to the set of parameters P, from the MLP (the weights and the biases).

The first thing to do is to calculate the derivative of the loss function with respect to each parameter of the MLP. This will be done by using the chain rule from calculus. So we have to start calculating the derivative of $e(\mathbf{y}, \mathbf{s}) \equiv e$ with respect to $\mathbf{y}$, for then calculating the derivative of e with respect to the parameters $w^{(2)}_{i,j}$, $b^{(2)}_j$, with which we will calculate the derivative from the previous layer parameters and so on until we reach the beginning of the network. This is called *backward pass*, as opposed to the forward pass which is just $MLP(\mathbf{x}) = \mathbf{y}$. So now we start the calculations:

Between layer 3 and layer 2

$$\frac{\partial e}{\partial y_j} = \frac{\partial}{\partial y_j}\left(\frac{1}{2}(s_1 - y_1)^2 + \frac{1}{2}(s_2 - y_2)^2\right) = -(s_j - y_j) \qquad \text{with } j = 1, 2$$

$$\frac{\partial y_j}{\partial w_{i,j}^{(2)}} = \frac{\partial}{\partial w_{i,j}^{(2)}} \left(\sigma\left(w_{1,j}^{(2)} a_1^{(2)} + w_{2,j}^{(2)} a_2^{(2)} + w_{3,j}^{(2)} a_3^{(2)} + w_{4,j}^{(2)} a_4^{(2)} + b_j\right)\right) =$$

$$= \sigma\left(w_{1,j}^{(2)} a_1^{(2)} + w_{2,j}^{(2)} a_2^{(2)} + w_{3,j}^{(2)} a_3^{(2)} + w_{4,j}^{(2)} a_4^{(2)} + b_j\right)\left(1 - \sigma\left(w_{1,j}^{(2)} a_1^{(2)} + w_{2,j}^{(2)} a_2^{(2)} + w_{3,j}^{(2)} a_3^{(2)} + w_{4,j}^{(2)} a_4^{(2)} + b_j\right)\right)$$

$$\frac{\partial}{\partial w_{i,j}^{(2)}}\left(w_{1,j}^{(2)} a_1^{(2)} + w_{2,j}^{(2)} a_2^{(2)} + w_{3,j}^{(2)} a_3^{(2)} + w_{4,j}^{(2)} a_4^{(2)} + b_j\right) = y_j(1 - y_j)(a_i^{(2)})$$

$$\boxed{\frac{\partial e}{\partial w_{i,j}^{(2)}} = \frac{\partial e}{\partial y_j} \cdot \frac{\partial y_j}{\partial w^{(2)_{i,j}}} = (y_j - s_j)\left(y_j(1 - y_j)(a_i^{(2)})\right)}$$

$$\frac{\partial y_j}{\partial b_j^{(2)}} = \frac{\partial}{\partial b_j^{(2)}} \left(\sigma\left(w_{1,j}^{(2)} a_1^{(2)} + w_{2,j}^{(2)} a_2^{(2)} + w_{3,j}^{(2)} a_3^{(2)} + w_{4,j}^{(2)} a_4^{(2)} + b_j\right)\right) =$$

$$= \sigma\left(w_{1,j}^{(2)} a_1^{(2)} + w_{2,j}^{(2)} a_2^{(2)} + w_{3,j}^{(2)} a_3^{(2)} + w_{4,j}^{(2)} a_4^{(2)} + b_j\right)\left(1 - \sigma\left(w_{1,j}^{(2)} a_1^{(2)} + w_{2,j}^{(2)} a_2^{(2)} + w_{3,j}^{(2)} a_3^{(2)} + w_{4,j}^{(2)} a_4^{(2)} + b_j\right)\right)$$

$$\frac{\partial}{\partial b_j^{(2)}}\left(w_{1,j}^{(2)} a_1^{(2)} + w_{2,j}^{(2)} a_2^{(2)} + w_{3,j}^{(2)} a_3^{(2)} + w_{4,j}^{(2)} a_4^{(2)} + b_j\right) = y_j(1 - y_j)$$

$$\boxed{\frac{\partial e}{\partial b_j^{(2)}} = \frac{\partial e}{\partial y_j} \cdot \frac{\partial y_j}{\partial b_j^{(2)}} = (y_j - s_j)\, y_j(1 - y_j)}$$

Defining $\delta^{(3)}$ as

$$\delta^{(3)} = (y_j - s_j)\, y_j(1 - y_j)$$

We have that:

$$\boxed{\frac{\partial e}{\partial w_{i,j}^{(2)}} = a_i^{(2)}\, \delta^{(3)}} \qquad \boxed{\frac{\partial e}{\partial b_j^{(2)}} = \delta^{(3)}}$$

Between layer 2 and layer 1

$$\frac{\partial a_j^{(2)}}{\partial w_{i,j}^{(1)}} = \frac{\partial}{\partial w_{i,j}^{(1)}} \left(\sigma\left(w_{1,j}^{(1)} a_1^{(1)} + w_{2,j}^{(1)} a_2^{(1)} + w_{3,j}^{(1)} a_3^{(1)} + w_{4,j}^{(1)} a_4^{(1)} + b_j\right)\right) = a_j^{(2)}(1 - a_j^{(2)})(a_i^{(1)})$$

$$\frac{\partial a_p^{(k+1)}}{\partial a_j^{(k)}} = \frac{\partial}{\partial a_j^{(k)}} \left(\sigma\left(w_{1,p}^{(k)} a_1^{(k)} + w_{2,p}^{(k)} a_2^{(k)} + ... + w_{m_k,p}^{(k)} a_{m_k}^{(k)} + b_p\right)\right) = a_p^{(k+1)}(1 - a_p^{(k+1)})(w_{j,p}^{(k)})$$

Noting that $y_p = a_p^{(3)}$

$$\frac{\partial e}{\partial w_{i,j}^{(1)}} = \sum_{p=1}^{2} \left(\frac{\partial e}{\partial y_p} \cdot \frac{\partial y_p}{\partial a_j^{(2)}} \cdot \frac{\partial a_j^{(2)}}{\partial w_{i,j}^{(1)}} \right) = \sum_{p=1}^{2} \left((y_p - s_p) \left(y_p(1-y_p) w_{j,p}^{(2)} \right) \left(a_j^{(2)}(1-a_j^{(2)})(a_j^{(1)}) \right) \right) =$$

$$= \left(a_j^{(2)}(1-a_j^{(2)})(a_j^{(1)}) \right) \sum_{p=1}^{2} \left((y_p - s_p) \left(y_p(1-y_p) w_{j,p}^{(2)} \right) \right)$$

$$\boxed{\frac{\partial e}{\partial w_{i,j}^{(1)}} = \left(a_j^{(2)}(1-a_j^{(2)})(a_j^{(1)}) \right) \sum_{p=1}^{2} \left((y_p - s_p) \left(y_p(1-y_p) w_{j,p}^{(2)} \right) \right)}$$

$$\frac{\partial e}{\partial b_j^{(1)}} = \sum_{p=1}^{2} \left(\frac{\partial e}{\partial y_p} \cdot \frac{\partial y_p}{\partial a_j^{(2)}} \cdot \frac{\partial a_j^{(2)}}{\partial b_j^{(1)}} \right) = \sum_{p=1}^{2} \left((y_p - s_p) \left(y_p(1-y_p) w_{j,p}^{(2)} \right) \left(a_j^{(2)}(1-a_j^{(2)}) \right) \right) =$$

$$= \left(a_j^{(2)}(1-a_j^{(2)}) \right) \sum_{p=1}^{2} \left((y_p - s_p) \left(y_p(1-y_p) w_{j,p}^{(2)} \right) \right)$$

$$\boxed{\frac{\partial e}{\partial b_j^{(1)}} = \left(a_j^{(2)}(1-a_j^{(2)}) \right) \sum_{p=1}^{2} \left((y_p - s_p) \left(y_p(1-y_p) w_{j,p}^{(2)} \right) \right)}$$

Defining $\delta^{(2)}$ as

$$\delta^{(2)} = \left(a_j^{(2)}(1-a_j^{(2)}) \right) \sum_{p=1}^{2} \left((y_p - s_p)\, y_p(1-y_p) w_{j,p}^{(2)} \right) = \left(a_j^{(2)}(1-a_j^{(2)}) \right) \sum_{p=1}^{2} w_{j,p}^{(2)} \delta^{(3)}$$

We have that:

$$\boxed{\frac{\partial e}{\partial w_{i,j}^{(1)}} = a_i^{(1)}\, \delta^{(2)}} \qquad \boxed{\frac{\partial e}{\partial b_j^{(1)}} = \delta^{(2)}}$$

Between layer 1 and layer 0 Noting that $\mathbf{x} = a^{(0)}$:

$$\frac{\partial a_j^{(1)}}{\partial w_{i,j}^{(0)}} = \frac{\partial}{\partial w_{i,j}^{(0)}} \left(\sigma \left(w_{1,j}^{(0)} x_1 + w_{2,j}^{(0)} x_2 + w_{3,j}^{(0)} x_3 + b_j \right) \right) = a_j^{(1)}(1-a_j^{(1)})(x_i)$$

$$\frac{\partial e}{\partial w_{i,j}^{(0)}} = \sum_{p=1}^{2}\left(\frac{\partial e}{\partial y_p}\sum_{q=1}^{4}\left(\frac{\partial y_p}{\partial a_q^{(2)}}\cdot\frac{\partial a_q^{(2)}}{\partial a_j^{(1)}}\cdot\frac{\partial a_j^{(1)}}{\partial w_{i,j}^{(0)}}\right)\right) =$$

$$= \sum_{p=1}^{2}\left(-(s_p-y_p)\sum_{q=1}^{4}\left(\left(y_p(1-y_p)w_{q,p}^{(2)}\right)\cdot\left(a_q^{(2)}(1-a_q^{(2)})w_{j,q}^{(1)}\right)\cdot\left(a_j^{(1)}(1-a_j^{(1)})x_i\right)\right)\right) =$$

$$= \left(a_j^{(1)}(1-a_j^{(1)})x_i\right)\sum_{p=1}^{2}\left(-(s_p-y_p)\sum_{q=1}^{4}\left(y_p\,(1-y_p)\,w_{q,p}^{(2)}\,a_q^{(2)}\,(1-a_q^{(2)})\,w_{j,q}^{(1)}\right)\right) =$$

$$= \left(a_j^{(1)}(1-a_j^{(1)})x_i\right)\sum_{p=1}^{2}\left(\sum_{q=1}^{4}(y_p-s_p)\,\left(y_p\,(1-y_p)\,w_{q,p}^{(2)}\,a_q^{(2)}\,(1-a_q^{(2)})\,w_{j,q}^{(1)}\right)\right) =$$

$$= \left(a_j^{(1)}(1-a_j^{(1)})x_i\right)\sum_{q=1}^{4}\left(\sum_{p=1}^{2}(y_p-s_p)\,\left(y_p\,(1-y_p)\,w_{q,p}^{(2)}\,a_q^{(2)}\,(1-a_q^{(2)})\,w_{j,q}^{(1)}\right)\right) =$$

$$= \left(a_j^{(1)}(1-a_j^{(1)})x_i\right)\sum_{q=1}^{4}\left(a_q^{(2)}\,(1-a_q^{(2)})\,w_{j,q}^{(1)}\sum_{p=1}^{2}(y_p-s_p)\,\left(y_p\,(1-y_p)\,w_{q,p}^{(2)}\right)\right)$$

$$\boxed{\frac{\partial e}{\partial w_{i,j}^{(0)}} = \left(a_j^{(1)}(1-a_j^{(1)})x_i\right)\sum_{q=1}^{4}\left(a_q^{(2)}\,(1-a_q^{(2)})\,w_{j,q}^{(1)}\sum_{p=1}^{2}(y_p-s_p)\,y_p\,(1-y_p)\,w_{q,p}^{(2)}\right)}$$

$$\frac{\partial a_j^{(1)}}{\partial b_j^{(0)}} = \frac{\partial}{\partial b_j^{(0)}}\left(\sigma\left(w_{1,j}^{(0)}x_1 + w_{2,j}^{(0)}x_2 + w_{3,j}^{(0)}x_3 + b_j\right)\right) = a_j^{(1)}(1-a_j^{(1)})$$

$$\frac{\partial e}{\partial b_j^{(0)}} = \sum_{p=1}^{2}\left(\frac{\partial e}{\partial y_p}\sum_{q=1}^{4}\left(\frac{\partial y_p}{\partial a_q^{(2)}}\cdot\frac{\partial a_q^{(2)}}{\partial a_j^{(1)}}\cdot\frac{\partial a_j^{(1)}}{\partial b_j^{(0)}}\right)\right)$$

By doing the same procedure as before, but changing the last derivative $\dfrac{\partial a_j^{(1)}}{\partial b_j^{(0)}}$ we arrive to:

$$\boxed{\frac{\partial e}{\partial b_j^{(0)}} = \left(a_j^{(1)}(1-a_j^{(1)}))\right)\sum_{q=1}^{4}\left(a_q^{(2)}\,(1-a_q^{(2)})\,w_{j,q}^{(1)}\sum_{p=1}^{2}(y_p-s_p)\,y_p\,(1-y_p)\,w_{q,p}^{(2)}\right)}$$

Defining $\delta^{(1)}$ as

$$\delta^{(1)} = \left(a_j^{(1)}(1-a_j^{(1)})\right) \sum_{q=1}^{4} \left(a_q^{(2)} \left(1-a_q^{(2)}\right) w_{j,q}^{(1)} \sum_{p=1}^{2} (y_p - s_p)\, y_p \left(1-y_p\right) w_{q,p}^{(2)} \right) =$$

$$= \left(a_j^{(1)}(1-a_j^{(1)})\right) \sum_{q=1}^{4} \left(a_q^{(2)} \left(1-a_q^{(2)}\right) w_{j,q}^{(1)} \sum_{p=1}^{2} w_{q,p}^{(2)}\, \delta^{(3)} \right) = \left(a_j^{(1)}(1-a_j^{(1)})\right) \sum_{q=1}^{4} \left(w_{j,q}^{(1)}\, \delta^{(2)} \right)$$

We have that:

$$\boxed{\frac{\partial e}{\partial w_{i,j}^{(0)}} = x_i\, \delta^{(1)}} \qquad\qquad \boxed{\frac{\partial e}{\partial b_j^{(0)}} = \delta^{(1)}}$$

With this information, we can update the parameters of the network using the procedure we have seen in section 3.1.1.

$$w \longrightarrow w - \alpha \frac{\partial e}{\partial w} \qquad\qquad b \longrightarrow b - \alpha \frac{\partial e}{\partial b}$$

Remember that α is the *learning rate* that tells us how much we change the parameters in the direction in which the function decreases every time we update them. Note that this is an iterative process, so once the parameters, P, from the network are updated, we repeat the process. Every iteration is called an *epoch*, and normally those are fixed.

Remark 10. *Note that as we are using the sigmoid function also in the last layer our outputs* $\boldsymbol{y}$ *will be between 0 and 1, so we expect the targets,* $\boldsymbol{s}$*, to follow this rule too. To accomplish that we can scale them by calculating the minimum and the maximum of each column and doing:*

$$\text{ValScaled} = \frac{\text{ValOriginal} - \text{MinVal}}{\text{MaxVal} - \text{MinVal}}$$

Then we can recover the original value by doing:

$$\text{ValOriginal} = \text{ValScaled}(\text{MaxVal} - \text{MinVal}) + \text{MinVal}$$

Although, the most common thing when dealing with regression problems is not to have the activation function in the last layer.

3.1.4 Adding complexity

The MLP studied in the previous section is very simple, so we can perform Back Propagation manually. But in practice, they have more neurons and layers. Moreover, there are a lot

of modifications that can be made to our basic setting. For example, one could change the optimization algorithm from gradient descent to another one, for example using stochastic methods. This is the case of one of the most commonly used optimization algorithms for neural networks, the *Adam optimizer* [4].

As mentioned we can also change the activation function. To integrate this change we would only need to modify the definition of δ to replace the derivative of the sigmoid activation function with the one that we are using. For example, if we want to use the sinusoidal function, $\sigma = sin(x)$, then we would replace the derivative by:

$$\sigma'(s) = sin'(x) = cos(x) = \sqrt{1 - sin^2(x)} = \sqrt{1 - \sigma^2(x)}$$

We have introduced the sinusoidal function as an activation function because it will be relevant in the following part of the text. The main idea of this section is to provide an understanding of what is happening at the basic level when we train a neural network. In the next sections and chapters, we will use more complex neural networks, losses, and activation functions, but it is all based on what has been discussed above.

3.2 Neural Fields

Definition 15. *A field is a quantity defined for all spatial and/or temporal coordinates.*

Those can be formulated then as functions that map spatial and/or temporal coordinates to $\mathbb{R}^n$.

Hence, using the universal approximation theorem, we approximate these fields using neural networks, giving rise to the term Neural Fields. So basically those are neural networks that take as input spatiotemporal coordinates and give as output a scalar that represents the value of the field in that spatiotemporal point.

In this setting, we can see the Signed Distance Functions of surfaces as fields. To each spatial coordinate where the SDF is defined, we assign its signed distance from the surface using the function 2.5. But we do not necessarily have to know what the value of the function is, especially in complex shapes. Here is where we will use neural networks to approximate the function 2.5.

In section 3.1.4 we have discussed how we can add complexity to the neural networks and how we train them. In this section, we will specify what changes are we making to the basic training scenario introduced before.

Remark 11. *Through this section, we will refer to the neural network used to approximate the SDF of the surface M, as F_θ. The subscript θ indicates the set of parameters of the neural network F. Then, the zero-isosurface of F_θ will be a surface M_{F_θ}, and we would like it to be as similar as possible to M so that we can say $M_{F_\theta} \equiv M$.*

The objective of this section is to show how we can get the neural network F_θ that approximates the SDF of M. To illustrate it better we will work with an example where our surface M will be the *Armadillo* [11] shown in figure 3.5. Note that the surface M is just conceptual, meaning that we do not have any parametrization, nor the SDF of this surface. In the next subsection, we will see how surfaces are normally represented in computer graphics.

Figure 3.5: Original Armadillo mesh

3.2.1 Data

To train our neural network we need data first. This data will be tuples of 3D points, $x \in \mathbb{R}^3$, and their corresponding value of the SDF $y = n(x) \in \mathbb{R}$. We will denote the set of these tuples as D, and it will be our dataset. But we often do not have this information, instead, we usually have two types of representations of the surface M. One is pointclouds, which are simply a set of points on the surface. These appear more often when working with sensors in computer vision and inverse graphics problems.

The other more common representation is meshes, which are a polygonal representation of the surface M. The most common polygonal meshes are the ones formed by triangles, called triangular meshes. So we do not have the surface M itself, but a discrete triangularization

of it. An example of this triangularization can be seen when zooming on the armadillo 3.6. We describe how to extract the dataset D from such meshes, but the process is analogous for pointclouds.

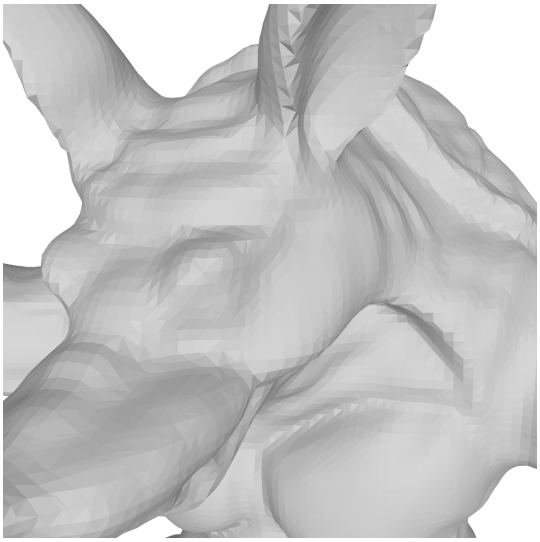

Figure 3.6: Zoom Armadillo mesh

To extract the dataset D, from a given mesh we will proceed as follows. First, we normalize the mesh to be in a 3D cube $[-1, 1]^3$. Then, we sample uniformly 5 million points inside this cube. For each one, we will calculate the distance to the closest polygon on the surface. Moreover, we have information on whether the point is inside or outside the shape, which gives us the sign. Now, we sample again 5 million points but this time we do not do it uniformly over the whole cube, but near the surface (i.e. the surface plus a Gaussian with a standard deviation of 0.1). We then compute the value for $n(x)$ of these points the same way we did before. an example of the process can be seen in figure 3.7.

Remark 12. *Note that in the formulation of the function $n(x)$ 2.5, we restricted the domain to be a small enough neighborhood so that no two normal rays passing through different points on the surface intersect in that neighborhood. We can check this to see if a point is valid or not and discard it in case it is not. Although, the results presented in figure 3.7 use more elaborated techniques that allow you to take more points.*

With this, we have our dataset D which is a sample of values of the SDF of M, ideally, 10 million points (some of them can be discarded). Note that, in reality, these values are not for the SDF of M, but for the SDF of the mesh. Nevertheless, this is a good enough approximation because our neural network will not have the information about the triangularization (it only sees the dataset of points D). Better results could be achieved if we use a very dense pointcloud instead of a mesh to represent M. In the rest of the section, we will refer to approximating the SDF of M for simplicity.

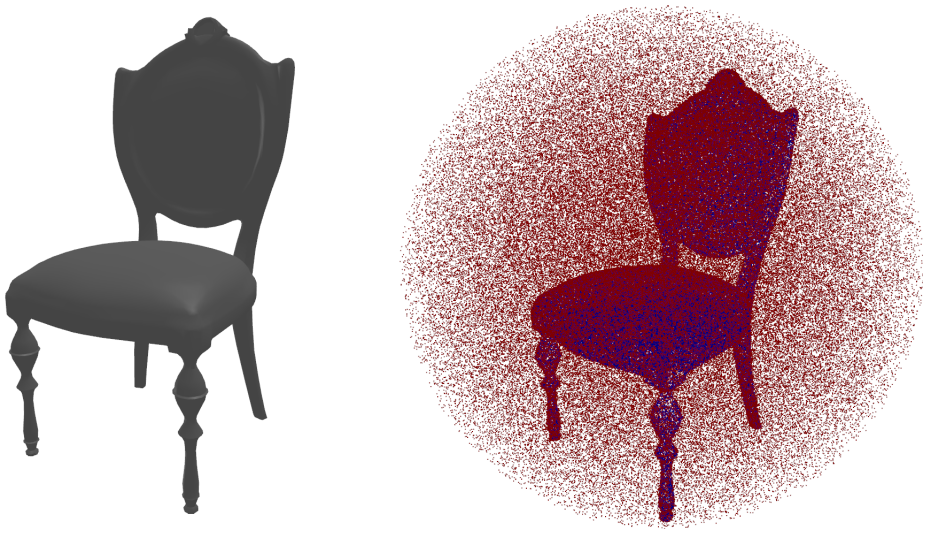

Figure 3.7: Result from sampling to get a dataset D. In the right image, the red points represent points with a positive sign (outside the surface) and the blue ones have a negative sign (inside the surface). Image from [13]

3.2.2 MLP Architecture and activation function

The **MLP architecture** is what defines the number of layers and how many neurons each of those has. In other applications, it also refers to how the connections between neurons are made, which gives rise to *Convolutional Neural Networks (CNN)*, *Recurrent Neural Networks (RNN)*, *Transformers*, or the more general ones, *Graph Neural Networks (GNN)*. But in our case, the simple MLP architecture presented in the section 3.1 will be enough.

The complexity of a neural network grows with the number of layers and/or with the number of neurons. Intuitively, a more complex neural network can represent more complex shapes. In the examples presented by Yang, Guandao et al. [17], the following dimensions (number of neurons per layer) were used depending on the complexity of the shape:

- Complex 3D shapes: 3-512-512-512-512-512-512-1.
- Simple 3D shapes: 3-512-512-512-1.
- 2D rectangle: 2-128-128-128-128-1.

Remark 13. *Note that the input dimensions are always the ones from the space we are in, and the output is always one (the signed distance predicted for that input).*

When dealing with neural implicit representations (not only for SDF, but also for images and

videos) the **sine activation function** has given better results than others [14]. In the case of SDF, the reason is that the loss function involves calculating the gradient of the neural network because we will be solving (approximately) the Eikonal equation 2.8. This makes architectures that use activations whose gradients are not well-behaved (such as ReLU or TanH) perform worse than the ones using the sine.

3.2.3 Loss and hyperparameters

The next thing to do is to define the loss function, keeping in mind that we have two objectives to fulfill by our neural network F: first, we want it to be a good approximation of the SDF of the surface M; second, we want it to be an SDF, and we can achieve it by enforcing that the Eikonal equation 2.8 is fulfilled. Remember that the tuples of our dataset are the input points $x \in \mathbb{R}^3$ and the ground truth SDF value, $y = n(x) \in \mathbb{R}$, with that we have that our loss function will be:

$$e(x, y) = (F_\theta(x) - y)^2 + \lambda_g \left(||\nabla_x F_\theta(x)|| - 1\right)^2 \tag{3.1}$$

The first term of the sum will ensure that the predicted values of the SDF are close to the ground truth. The second is the one that ensures the Eikonal equation is fulfilled and is derived by subtracting one from both sides in the equation 2.8. The constant λ_g is just a hyperparameter that weights the importance of the second term for the optimization process. In practice, it was set to $\lambda_g = 0.01$ according to Gropp et al. [6]. By applying the backpropagation algorithm with this loss function we will decrease this error, therefore F_θ will be a better approximation of the SDF of M. Remember that we are working now with a much larger neural network, so we do not compute all the derivatives required manually. Instead, we can use computational tools and frameworks that provide us with automatic differentiation capabilities. Those basically store a computational graph of the operations performed to easily calculate the derivatives with respect to each weight in the network.

In the loss function appears the following term $\nabla_x F_\theta(x)$. This means calculating the gradient of F_θ with respect to the inputs x. To do this we can sample a number of points from the cube $[-1, 1]^3$, process them with the neural network, and compute the derivative of the output with respect to the sampled point (using the automatic differentiation). Note that there is no loss function involved in this process.

One last remark is that, for the optimization process, the Adam optimizer [4] was used, instead of the basic Gradient Descent described in the previous section. The network was trained for 300000 steps (iterations of the backpropagation algorithm) with a learning rate of 1e - 5.

Chapter 4

Shape smoothing and sharpening

In this chapter, we assume that we have a neural network F representing a 3D surface (neural implicit representation), M_F. We can obtain this neural network following what we have done in the previous chapter. Note that in this chapter we denote the network as F without the subscripts θ. This is because we are no longer changing the parameters of this network, we will assume that it is already a good representation of the surface. So now F can be seen as a static function that takes as input a 3D point in the space and outputs the SDF with respect to the surface M_F. Then, F is a (good) approximation of the true SDF, n.

4.1 Objective

Our objective in this chapter is to obtain another neural implicit representation of the surface M_F but after with a smoothing or sharpening (we will refer to both as filtering) effect. We could do it in two ways:

1. Apply some filtering (smoothing or sharpening) algorithm to the original surface M and train from scratch a neural network as we saw in the previous chapter.

2. Manipulating the neural network F to obtain another neural network H that represents the surface M after the filtering process.

We are interested in the second way to do it because it seems more natural and direct, as well as more interesting for future applications.

4.2 Smoothing and sharpening

We will use another neural network, G_θ, with the same architecture and activation function as F. Moreover, we will initialize the weights and biases, θ, from H_θ to be the same as the already trained parameter of F. We will call the shape represented by H_θ, M_{H_θ}. Note that at the beginning of the process $M_F = M_{G_\theta}$, but at the end of the process, M_{G_θ} has to represent the same shape as M_F after a smoothing or sharpening process.

We want the neural network G_θ to fulfill three properties at the end of the filtering process:

1. It has to preserve the original shape represented by F, G_F.
2. It has to be enforced to remain an SDF (i.e. fulfilling the Eikonal equation 2.8).
3. The curvature of the surface represented, M_{H_θ}, has to change. It will decrease in the case of smoothing filtering and increase in the case of sharpening.

To convert the network G_θ into the desired one we will use the backpropagation algorithm to change the weights θ so that the above properties are fulfilled. A way to introduce these restrictions is in the loss function. Note that this time we do not need a ground truth y, which in the Machine Learning community is referred to as unsupervised learning (although self-supervised would be more accurate in this case). So the loss function will only depend on the input x.

$$e(x) = (G_\theta(x) - F(x))^2 + \lambda_g \left(||\nabla_x G_\theta(x)|| - 1\right)^2 + \lambda_k(\overline{k}_{G_\theta}(x) - \beta\, \overline{k}_F(x))^2$$

We will explain each term in detail:

1. $(G_\theta(x) - F(x))^2$: This enforces the shape M_{H_θ} to be the same as M_F. We do not use the ground truth y from the sampled dataset D (as in the previous section), because this is limited to a few points (approximately 10 million) in $[-1, 1]^3$. Contrary, using $F(x)$ allow us to take any $x \in [-1, 1]^3$.
2. $\lambda_g \left(||\nabla_x G_\theta(x)|| - 1\right)^2$: This term is the Eikonal 2.8 regularization term that enforces the representation to remain an SDF. The constant λ_g weights the importance given to this restriction.
3. $\lambda_k(\overline{k}_{G_\theta}(x) - \beta\, \overline{k}_F(x))^2$: This term makes the mean curvature of each level set from the new network, $\overline{k}_{G_\theta}(x)$ be proportional to the original one $\overline{k}_F(x)$ by a factor of β. If $\beta < 1$ then the curvature will decrease, smoothing the surface. If $\beta > 1$ then the curvature will increase, sharpening the surface. The constant λ_k weights the importance given to this restriction.

We can see different experiments and their results in figures 4.1 for smoothing and 4.2 for sharpening.

(a) $\beta = 1$ (original mesh) (b) $\beta = 0.6$ (c) $\beta = 0.0$

Figure 4.1: Shape smoothing

(a) $\beta = 1$ (original mesh) (b) $\beta = 2$ (c) $\beta = 10$

Figure 4.2: Shape sharpening

Before explaining how we calculate the terms $\overline{k}_F(x)$ and $\overline{k}_{G_\theta}(x)$ we will introduce a definition that will be useful.

Definition 16. ***Divergence of a vector field in Cartesian Coordinates*** *Let $G : \mathbb{R}^3 \to \mathbb{R}^3$, $G(x, y, z) = (G_1(x, y, z), G_2(x, y, z), G_3(x, y, z))$ be a vector field in Cartesian coordinates. We define the divergence of that field, $div(G)$, as:*

$$div(G) = \nabla \cdot G = \left(\frac{dG_1}{dx} + \frac{dG_2}{dy} + \frac{dG_3}{dz} \right)$$

Remark 14. ***How the mean curvature is calculated:*** *We will explain how to calculate the normal curvature on the point x with respect to F, but is analogous for G_θ*

Remember that we defined the shape operator as:

$$W_p := -dN_p : T_pS \to T_pS$$

Then we had that the mean curvature in a point p on the surface was, because 7:

$$\overline{k}(p) := \frac{tr(W_p)}{2} = \frac{tr(-H_n)}{2} = -\frac{tr(H_n)}{2}$$

Nevertheless, in the implementation of Guandao Yang et al. [17], the shape operator is defined without using the negative sign, $W_p := dN_p$. That will make the normal curvature be $\overline{k}(p) := \frac{tr(W_p)}{2} = \frac{tr(H_n)}{2}$. Note that this change in the sign is not relevant because the sign of the mean curvature depends on the sense of the normal we are choosing, which depends on the parametrization we are choosing. For simplicity in the calculations we will follow the convention taken in [17], so we will take $\overline{k}(p) = \frac{tr(H_n)}{2}$

Our point p is called x, and the SDF n is F.Then we have that $\overline{k}(x) = \frac{tr(H_F)}{2}$. Remember that the Hessian matrix looks as follows:

$$H_F = \begin{pmatrix} \frac{d^2F}{dx^2} & \frac{d^2F}{dxdy} & \frac{d^2F}{dxdz} \\ \frac{d^2F}{dydx} & \frac{d^2F}{dy^2} & \frac{d^2F}{dydz} \\ \frac{d^2F}{dzdx} & \frac{d^2F}{dzdy} & \frac{d^2F}{dz^2} \end{pmatrix}$$

$$\overline{k}(x) = \frac{tr(H_F)}{2} = \frac{d^2F}{dx^2} + \frac{d^2F}{dy^2} + \frac{d^2F}{dz^2}$$

So we would need to calculate the Hessian matrix of F and then calculate the trace. This can be more simply implemented by using the divergence, as done in the implementation of Guandao Yang et al. [17]. If we take the vector field $\nabla F = (\frac{dF}{dx}, \frac{dF}{dy}, \frac{dF}{dz})$ we can calculate its divergence as:

$$div(\nabla F) = \nabla \cdot (\nabla F) = \frac{d^2F}{dx^2} + \frac{d^2F}{dy^2} + \frac{d^2F}{dz^2} = \overline{k}(x)$$

This way we only need to calculate ∇F using the automatic differentiation tools mentioned in the previous section and calculate its divergence with respect to the inputs (using the automatic differentiation tools again)

After the training process of the neural network G_θ with the error metric introduced here, we expect it to represent the same shape after the filtering operation desired.

4.3 Limitations and research directions

- **Speed**: The shape smoothing and sharpening operation as we have described it implies an optimization process by training the neural network. This is time consuming and does not allow interactivity during the editing. One possible solution could be trying to accelerate the training, by reducing the number of steps needed or making them faster. But this approach would not give the orders of magnitude of improvement that are needed. Another alternative is to use what are known as *hypernetworks*. In a generic sense, this are neural networks which output the weights of other neural networks. We can use the weights of the original neural field as input and train the network to perform the smoothing operation (in the weight space) and return the weights of the smoothed SDF. This would reduce the iterative process of training the network to a single forward pass. Similar approaches have shown success in consistent style transfer with NeRF (Neural Radiance Fields) [2], where artistic styles are applied to 3D scenes consistently 4.3.

Figure 4.3: (a) First the model learns an implicit representation of a 3D scene that disentangles the geometry and appearance. (b) Then, the style information is transfered using the hypernetwork. (c) Finally, we can render stylized novel views with a consistent appearance at various view angles. Image from [2]

- **Appearance**: We have discussed neural fields as representation of Signed Distance Functions, but those can be used for representing other fields. Specifically, we can encode the radiance field of an scene, which gives raise to one of the most famous application of neural fields, NeRF (Neural Radiance Fields) [12]. Those can be used to encode the appearance of a 3D scene, including lighting, reflections etc. Nevertheless, their performance is not as good for reconstructing the geometry of that scene. By combining SDF representation with NeRF we can achieve accurate geometry representations together with the appearance [1]. Moreover, once we have a way to edit the SDF, as we have shown in this work,

we can perform this kind of operations more robustly over the entire scene (including the appearance).

- **Compositing with Deep Learning pipelines**: One of the advantages of representing scenes as neural fields is using Deep Leaning techniques to those. We have already mentioned style transfer, but we can go beyond that. For example, we can learn the distribution of natural objects to get generative models of SDF [18] [3]. Or we can combine NeRF with the latest advances in Natural Leanguage Processing (NLP) to detect and segment objects in the 3D scene, based on queries in natural language 4.4[10].

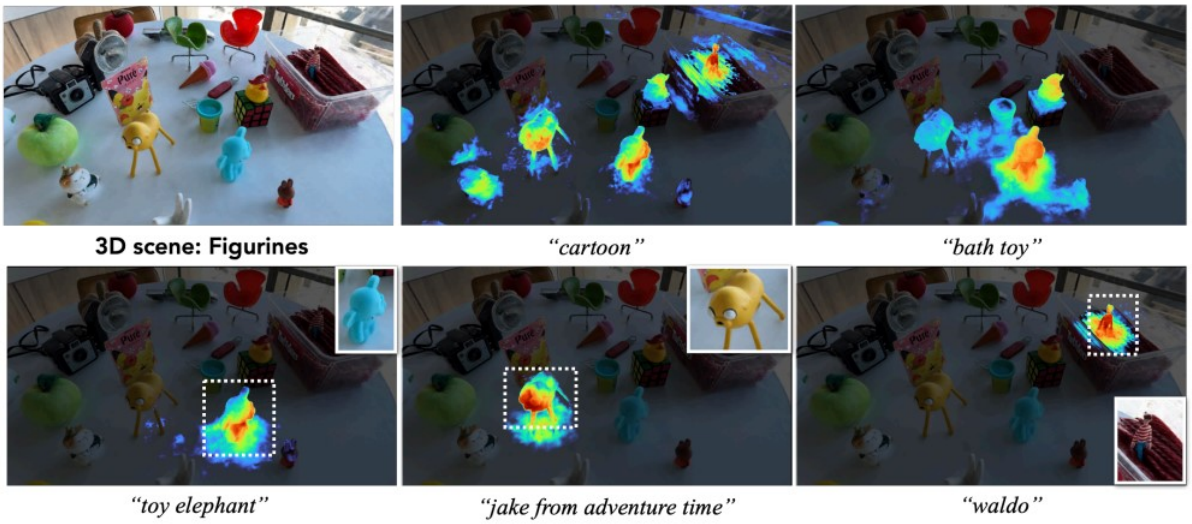

Figure 4.4: Multiple queries from objects in the scene using natural language. The highlighted regions correspond to the attention mask of the language model, which is used as a relevancy mask. Note that the input to the model is the natural language phrase that appears below each image. Image from [10]

Chapter 5

Conclusions

In conclusion, this work has provided a comprehensive overview of how 3D shapes can be represented using Signed Distance Functions approximated by neural networks. We have explained fundamental geometry concepts and their application to these functions, and we have delved into the workings of neural networks as function approximators. Our work has culminated in an explanation of how neural networks can be used to represent Signed Distance Functions of 3D shapes and how shape smoothing and sharpening can be performed on them.

Our objective was to provide a solid foundation for further exploration of this topic, with a focus on providing a mathematical explanation of these techniques, which we believe is essential for the continued development of the field.

While our work has focused on shape filtering as an example of geometry processing operations, there is still much to be explored in this area. Future research may consider other types of operations and their applications, such as shape deformation [17] or feature extraction. Additionally, it is also interesting to explore how to connect this kind of representation with other deep learning techniques. We consider that it is also essential to further investigate more efficient ways to create and manipulate these representations.

Overall, we hope that this work serves as a valuable resource for those interested in exploring the representation of 3D shapes using Signed Distance Functions and neural networks, and that it contributes to the continued growth and development of this field.

Bibliography

[1] Dejan Azinović, Ricardo Martin-Brualla, Dan B Goldman, Matthias Nießner, and Justus Thies. Neural rgb-d surface reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6290–6301, June 2022.

[2] Pei-Ze Chiang, Meng-Shiun Tsai, Hung-Yu Tseng, Wei-Sheng Lai, and Wei-Chen Chiu. Stylizing 3d scene via implicit representation and hypernetwork. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, January 2022.

[3] Gene Chou, Yuval Bahat, and Felix Heide. Diffusion-sdf: Conditional generative modeling of signed distance functions. *arXiv preprint arXiv:2211.13757*, 2022.

[4] Jimmy Ba Diederik P. Kingma. Adam: A method for stochastic optimization, 2014. Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[6] Amos Gropp, Lior Yariv, Niv Haim, Matan Atzmon, and Yaron Lipman. Implicit geometric regularization for learning shapes. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 3789–3799. PMLR, 13–18 Jul 2020.

[7] Juha Heinonen. Lectures on lipschitz analysis introduction, 2004. `http://www.math.jyu.fi/research/reports/rep100.pdf`.

[8] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[9] N. Johnson, P. Vulimiri, A. To, X. Zhang, C. Brice, Branden Kappes, and Aaron Stebner. Machine learning for materials developments in metals additive manufacturing, 05 2020.

[10] Justin Kerr, Chung Min Kim, Ken Goldberg, Angjoo Kanazawa, and Matthew Tancik. Lerf: Language embedded radiance fields. *arXiv preprint arXiv:2303.09553*, 2023.

[11] Venkat Krishnamurthy and Marc Levoy. Fitting smooth surfaces to dense polygon meshes. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, page 313–324, New York, NY, USA, 1996. Association for Computing Machinery.

[12] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020.

[13] Izaak Neutelings. Neural networks latex. TikZ.net Graphics with TikZ in LaTeX, May 2022. `https://tikz.net/neural_networks/`.

[14] Vincent Sitzmann, Julien N.P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. In *Proc. NeurIPS*, 2020.

[15] Reactant (`https://math.stackexchange.com/users/250971/reactant`). How to derive the 3d equation of a torus? Mathematics Stack Exchange. URL:`https://math.stackexchange.com/q/1352920` (version: 2020-06-12).

[16] Yiheng Xie, Towaki Takikawa, Shunsuke Saito, Or Litany, Shiqin Yan, Numair Khan, Federico Tombari, James Tompkin, Vincent Sitzmann, and Srinath Sridhar. Neural fields in visual computing and beyond. *Computer Graphics Forum*, 2022.

[17] Guandao Yang, Serge Belongie, Bharath Hariharan, and Vladlen Koltun. Geometry processing with neural fields. In *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021.

[18] Xin-Yang Zheng, Yang Liu, Peng-Shuai Wang, and Xin Tong. Sdf-stylegan: Implicit sdf-based stylegan for 3d shape generation. In *Comput. Graph. Forum (SGP)*, 2022.