UNIVERSITAT DE BARCELONA

Treball final de grau

GRAU DE ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

Optimizing Deep Learning Models using Topological Descriptors

Author: Oriol Saguillo González

Director:	**Dr. Sergio Escalera and Rubén Ballester**
Realized at:	**Department of Computer Science and Mathematics**
Barcelona,	**12 de juny de 2022**

Contents

Abstract

The combination of persistent homology and deep learning has been used frequently in recent years in the state of the art. Some of the most important works in this combination are about explaining the behaviour of neural networks or the application to improve the segmentation of elements in images. This project studies an application of persistent homology, a tool that describes holes of a point cloud, which tries to optimize the topological structure of the neural network in the training process.

Persistent homology is the most widely used mechanism in topological data analysis. Its main objective is to focus on the description of holes in a point cloud through a persistence diagram, this being a collection of these topological holes. Therefore, a topological descriptor can be obtained with such a diagram; a function whose main objective is to get a value that represents the complexity of the point cloud. For example, if a point cloud has multiple holes, a topological descriptor can get a number that represents them.

Deep learning has been a cutting-edge field for the last decades. It can be set as the representation of the human brain computationally, which is the most potent model in artificial intelligence. But, in all deep learning and machine learning problems attempted (image classification, sentiment analysis, etc.), overfitting will always be a significant threat during model training. It remains a problem that currently has no solution.

The proof-of-concept optimization framework of the project is to use differentiable topology and to show the viability of distinct topological descriptors that can optimize the network. Then, this could be used towards regularizing the learning of a network to improve its performance. This framework will become the first step towards my own attempt at avoiding the problem of overfitting and making networks learn rather than memorizing the training dataset.

Resum

La combinació d'homologia persistent i aprenentatge profund s'ha utilitzat amb freqüència en els últims anys en l'estat de l'art. Algunes de les obres més importants en aquesta combinació són sobre l'explicació del comportament de les xarxes neuronals o l'aplicació per millorar la segmentació d'elements en les imatges. Aquest projecte estudia una aplicació d'homologia persistent, una eina que descriu els forats d'un núvol de punts, que intenta optimitzar l'estructura topològica de la xarxa neuronal en el procés d'entrenament.

L'homologia persistent és el mecanisme més utilitzat en l'anàlisi topològica de dades. El seu objectiu principal és centrar-se en la descripció dels forats en un núvol de punts a través d'un diagrama de persistència, sent aquesta una col·lecció d'aquests forats topològics. Per tant, es pot obtenir un descriptor topològic amb aquest diagrama; una funció l'objectiu principal de la qual és obtenir un valor que representa la complexitat del núvol de punts. Per exemple, si un núvol de punts té múltiples forats, un descriptor topològic pot obtenir un nombre que els representa.

L'aprenentatge profund ha estat un camp d'avantguarda durant les últimes dècades. Es pot establir com la representació del cervell humà, que és el model més potent de la intel·ligència artificial. Però, en tots els problemes d'aprenentatge profund i d'aprenentatge automàtic intentats (classificació d'imatge, anàlisi de sentiment, etc.), l'adaptació excessiva sempre serà una amenaça significativa durant l'entrenament de models. Continua sent un problema que actualment no té solució.

El *framework* de la prova del concepte del projecte és utilitzar la topologia diferenciable i mostrar la viabilitat de diferents descriptors topològics que poden optimitzar la xarxa. Llavors, això es podria utilitzar per regularitzar l'aprenentatge d'una xarxa per millorar el seu rendiment. Aquest *framework* es convertirà en el primer pas cap al meu propi intent d'evitar el problema del *overfitting* i l'aprenentatge de les xarxes en lloc de memoritzar el conjunt de dades de formació.

Resumen

La combinación de homología persistente y aprendizaje profundo se ha utilizado con frecuencia en los últimos años en el estado del arte. Algunas de las obras más importantes en esta combinación son sobre la explicación del comportamiento de las redes neuronales o la aplicación para mejorar la segmentación de elementos en las imágenes. Este proyecto estudia una aplicación de homología persistente, una herramienta que describe los agujeros de una nube de puntos, que intenta optimizar la estructura topológica de la red neuronal en el proceso de entrenamiento.

La homología persistente es el mecanismo más utilizado en el análisis topológico de datos. Su objetivo principal es centrarse en la descripción de los agujeros en una nube de puntos a través de un diagrama de persistencia, siente esta una colección de estos agujeros topológicos. Por lo tanto, se puede obtener un descriptor topológico con este diagrama; una función el objetivo principal de la cual es obtener un valor que representa la complejidad de la nube de puntos. Por ejemplo, si una nube de puntos tiene múltiples agujeros, un descriptor topológico puede obtener un número que los representa.

El aprendizaje profundo ha sido un campo de vanguardia durante las últimas décadas. Se puede establecer como la representación del cerebro humano, que es el modelo más potente de la inteligencia artificial. Pero, en todos los problemas de aprendizaje profundo y de aprendizaje automático intentados (clasificación de imagen, análisis de sentimiento, etc.), la adaptación excesiva siempre será una amenaza significativa durante el entrenamiento de modelos. Continúa siendo un problema que actualmente no tiene solución.

El *framework* de la prueba del concepto del proyecto es utilizar la topología diferenciable y mostrar la viabilidad de diferentes descriptores topológicos que pueden optimizar la red. Entonces, esto se podría utilizar para regularizar el aprendizaje de una red para mejorar su rendimiento. Este *framework* se convertirá en el primer paso hacia mi propio intento de evitar el problema del *overfitting* y el aprendizaje de las redes en lugar de memorizar el conjunto de datos de formación.

Chapter 1

Introduction

Discrete mathematics is the study of mathematical structures that can be represented in integer format (discrete) instead of in a real format (continuous). The research in this area increased in the latter half of the twentieth century, partly due to the development of digital computers, which operate in "discrete" steps and store data in "discrete" bits. Concepts and notations from the field are useful in studying and describing objects and problems in branches of computer science. The subfield of interest for this project is computational topology with Topological Data Analysis, which develops topological techniques for robust analysis of scientific data.

The research growth in Topological Data Analysis area is the motivation of this project. Or, in more specific terms, understanding what persistent homology is and how to apply the tool in the Deep Learning field. I thought of this project as an opportunity to innovate and contribute to the scientific community.

Persistent homology is a computational method, for finding features of a simplicial complex in different filtrations. But what are simplicial complexes and filtrations? First of all, to define what a simplicial complex S is, we need to have point cloud P. To construct the simplicial complex S of the point cloud P, we have to imagine the points as the centre of circumferences with the same radius. The grouping of interactions among these circumferences, or the absence of it, can result in various S that need to be studied. In simpler words, depending on the intersection between the circumferences, multiple geometric structures will be defined in this S. As an example, some of these structures can be: points (No intersection with other points), line segments (one intersection between two points), triangles (three intersections among three points) and more of these geometry constructions, depending on the intersection among the balls defined by the points, or other condition previously defined. All in all, S is the set of all of those geometric structures. Furthermore, filtration is the concatenation of simplicial complexes S from the same point cloud P. This concatenation of different S is achieved for different radius values, $r \in [0, \infty)$.

However, why can this tool be useful in computer science, especially in Artificial Intelligence? There have been several scientists trying to implement persistent homology in machine learning classifiers to solve the overfitting problematic. The overfitting issue in machine learning modelling occurs when the creation of a function that extrapolates the input data does not generalize well. In consequence, when new data arrives, this function performs poorly in comparison with the training data. As a first attempt, Chao Chen and others proposed the following regularizer in 2018: *"A Topological Regularizer for Classifiers via Persistent Homology"* [8]. After this first approach to avoid the mentioned issue, several investigations tried to implement this topological regularizer [4, 15].

The chosen models to be combined with persistent homology optimization are Deep Neural Networks, which are models that try to replicate humans' brain functionality. To make this union possible, Ballester et al., *"Towards explaining the generalization gap in neural networks using topological data analysis"* work will be used, in which a pipeline has been developed to get persistence diagrams from computational graphs. These diagrams are a collection of information on topological features, obtained with persistent homology performed on computational graphs. Finally, it has been demonstrated, in some cases, that overfitting can be explained with the aforementioned Persistent Homology, achieving great results in the state-of-the-art. [3]

The purpose of this work is to, given a persistence diagram from a Deep Neural Network, get a personalized topological optimization, the first step to achieve a personalized topological regularizer. To be able to achieve this optimization, checking if the differentiability of persistence diagrams is possible is needed first. There has been explanations of the differentiability of a point cloud given its persistence diagram [11]. Later some Oxford researchers came with a more general framework in [20]. In these works, it has been shown that the differentiability of a given point cloud can be performed from the persistence diagram generated from it. The idea will be to differentiate these persistence diagrams from the topological descriptors, which are scalars generated from the persistence diagram. Two descriptors are used for this project and each of them will try to achieve different topologies during the optimization in the neural network. As this project will work only with the Vietoris-Rips filtration, differentiability will be explained from this filtration. This filtration will be detailed in theory.

The novelty proposed in this bachelor's thesis is the combination of the differentiability of persistence diagrams on the representation of neural networks to vector spaces. Therefore, our aim will be building a framework which optimizes neural networks based on its topology. Neural networks are going to be represented by the space defined in [3]. By the end of this thesis, the potential of this framework will be discussed and how it could become a solution for overfitting which, currently, does not have an automatic solution.

Report Organization

In chapter 2, all the theoretical content, which is needed to understand this project, can be found. For instance, section 2.1 contains the mathematical background to understand topological data analysis. Furthermore, the differentiability of persistence diagrams and topological descriptors is explained. Therefore, all the maths-related content will be provided in this section. Next, the basics of Deep Learning will be described alongside the pipeline to get a point cloud from the deep learning model.

Chapter 3 explains the development of the framework, as well as the processes required to perform the topological optimization. The datasets and the different architectures used to validate the proper functioning will also be detailed. And finally, the different subprocesses to perform the point cloud optimization will be discussed.

In chapter 4 you will find the results of the experiments and an interpretation of the results produced. At the end of each section, the results will be judged as to whether they have fulfilled the project's objectives.

Finally, in Chapter 5 the conclusions of the project will be drawn, as well as possible new possibilities for this model and how it can evolve.

Chapter 2

Theory

2.1 Topological Data Analysis

This section explains the basis of the concepts applied for the project related to Topological Data Analysis. To achieve this the homology will be explained among other concepts. For more information, I extracted the major part of the information from: *"Computational Topology An Introduction"* [10].

2.1.1 Complexes

There are many ways to represent a point cloud, this being a set of data points in a space, one of them being a decomposition into simple pieces. It is qualified as a complex if the pieces are topologically simple and their common intersections are lower-dimensional pieces of the same kind.

Let $u_0, u_1, ..., u_k$ be points in $\mathbb{R}^d$. A point $x = \sum_{i=0}^{k} \lambda_i u_i$ is an *affine combination* of the u_i if the λ_i sum to 1. If the $k+1$ points are *affinely independent*, by which we mean that any two affine combinations, $x = \sum \lambda_i u_i$ and $y = \sum \nu_i u_i$, are the same if only if $\lambda_i = \nu_i$ for all i. The $k+1$ points are affinely independent if only if the k vectors $u_1 - u_0$, for $1 \leq i \leq k$, are linearly independent. An affine combination, $x = \sum \lambda_i u_i$, is a *convex combination*, if all λ_i are non-negative. The *convex hull* is the set of convex combinations.2.1

Definition 2.1 (k-simplex). *A k-simplex is the convex hull of $k+1$ affinely independent points, $\sigma = conv\{u_0, u_1, .., u_k\}$.*

For some dimensions special names are used, *vertex* for 0-simplex, *edge* for 1-simplex, *triangle* for 2-simplex, and *tetrahedron* for 3-simplex. A graphical example can be seen at 2.2.

Definition 2.2 (Face). *A face of σ is the convex hull of a non-empty subset of the u_i and it is proper if the subset is not the entire set.*

Sometimes $\tau \leq \sigma$ is written if τ is a face and $\tau < \sigma$ if it is a proper face of σ. The *boundary* of σ, denoted as bdσ, is the union of all proper faces, and the *interior* is everything

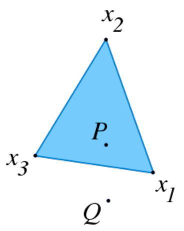


Figure 2.1: Given three points x_1, x_2, x_3 in a plane as shown in the figure, the point P is a convex combination of the three points, while Q is not. Q is however an affine combination of the three points.

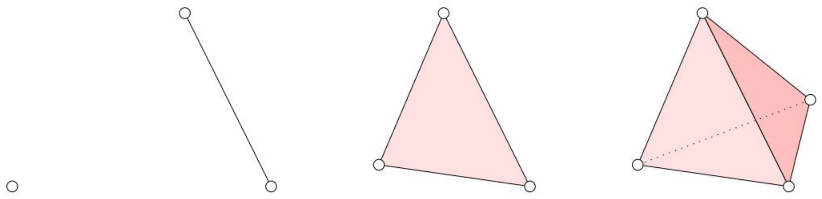


Figure 2.2: From left to right: a vertex, an edge, a triangle, and a tetrahedron. We note that an edge has two vertices, a triangle has three edges, and a tetrahedron has four triangles as faces.[10]

else, int$\sigma = \sigma -$ bdσ. A point $x \in \sigma$ belongs to intσ if only if all the coefficients λ_i are positive.

Definition 2.3 (Simplicial Complex)**.** *A simplicial complex is a finite collection of simplices K such that $\sigma \in K$ and $\tau \leq \sigma$ implies $\tau \in K$, and σ, $\sigma_0 \in K$ implies $\sigma \cap \sigma_0$ is either empty or a face of both.*

The dimension of K is the maximum dimension of any of its simplices. The underlying space denoted as $|K|$, is the union of its simplices together with the topology inherited from the ambient Euclidean space in which the simplices live.2.3

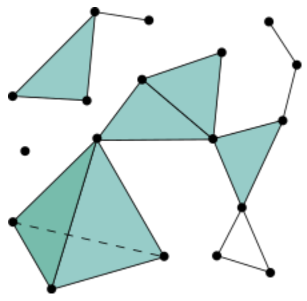


Figure 2.3: An example of a simplicial complex. It consists of 17 points (0-simplex), 22 edges (1-simplex), 8 triangles (2-simplex) and 1 tetrahedron (3-simplex).

2.1.2 Abstract Simplicial Complex

First, a definition of the data structure used in topological data analysis will be given.

Definition 2.4. *An abstract simplicial complex is a finite collection of sets A such that $\alpha \in A$ and $\beta \subseteq \alpha$ implies $\beta \in A$*

To understand better this concept, we can visualize an example in [28]:

Example 2.5. [Chess-board Complex] Let V be the positions on a chess board. $\sum$ collects position subsets of V where one can place queens without capturing each other.

If $\sigma \in \sum$ is a set of "safe" positions, then any subset $\tau \subseteq \sigma$ is also a set of "safe" positions.

2.1.3 Vietoris-Rips Complex

In order to generate the simplicial complexes algorithmically, the points need to be thought of as the center of "balls" and will be defined with the same radius in all of them. Now it is easier to define how it is formed a Vietoris-Rips complex. As an example, we can imagine this complex in an Euclidean space $\mathbb{R}^2$, which leads to think of the "balls" as circles.

Before defining the Vietoris-Rips complex, first we need to define the diameter of a simplicial complex:

Definition 2.6 (Diameter of a set)**.** *The diameter of a set in Euclidean space is the supremum over the distances between its points. Since the simplices of K are point sets in Euclidean space, their diameters are well defined.*

The Vietoris-Rips complex can be defined as follows:

Definition 2.7. *The Vietoris-Rips Complex of a finite set S of points in $\mathbb{R}^d$, and a radius r, it is formed by all subsets of diameter at most 2r.*

$$\text{Vietoris-Rips}(r) = \{\sigma \in S | \text{diam } \sigma \leq 2r\}$$

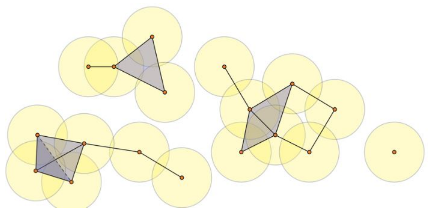


Figure 2.4: Example of a Vietoris-Rips complex. The 18 points are 0-simplices. Two 0-simplices form a 1-simplex (an edge) if their $\epsilon/2$-neighbourhoods (yellow circles) intersect. Three vertices form a 2-simplex (a triangle) if they are pairwise connected by edges. Four vertices form a 3-simplex (a tetrahedron) if they are pairwise connected by edges. [26]

2.1.4 Homology

Homology is a mathematical formalism for talking in a quantitative and unambiguous manner about how a space is connected. It provides a mathematical language for the holes in a point cloud. Its main ingredients are group operations and maps that relate topologically meaningful subsets of a space with each other.

Before starting with the construction of what homology is, lets define what is a group and the quotient space.

Definition 2.8 (Group[6]). *A group is a set G together with a binary operation on G, here denoted* $'' \cdot ''$*, that using the operation with 2 elements* $a, b \in G$*, such that the following three requirements, known as axioms of group, are satisfied*

- ***Associativity****: For all* $a, b, c \in G$*, one has* $(ab)c = a(bc)$*.*
- ***Identity element****: There exists and element* $e \in G$ *such that, for every* $a \in G$*, one has* $ea = a$ *and* $ae = a$*. Such element is unique, It is called the identity element of the group.*
- ***Inverse element****: For each* $a \in G$ *there exists and element* $b \in G$ *such that* $ab = e$ *and* $ba = e$*, where e is the identity element. For each a, the element b is unique; it is called the inverse of a and is commonly denoted as* a^{-1}*.*

To understand better the definition of what a group is, the following example is given:

Example 2.9. Given the set of integers, $\mathbb{Z} = \{.., -2, -1, 0, 1, 2, ...\}$ and the addition operation form a group. Given $a, b \in \mathbb{Z}$ the sum $a + b \in \mathbb{Z}$. This says that $+$ is a binary operation on $\mathbb{Z}$. [6]

Definition 2.10 (Quotient Space[7]). *Let V be a vector space over a field K, and let N be a subspace of V. We define an equivalence relation* $\sim$ *on V by stating that* $x \sim y$ *if* x - $y \in N$*. That is, x is related to y if one can be obtained from the other by adding an element of N. From this definition, one can deduce that any element of N is related to the zero vector; more precisely, all the vectors in N get mapped into the equivalence class of the zero vector.*

To understand better the definition of what a quotient space is, the following example is given:

Example 2.11. Let $X = R^2$ be the standard Cartesian plane, and let Y be a line through the origin in X. Then the quotient space X/Y can be identified with the space of all lines in X which are parallel to Y. That is to say that, the elements of the set X/Y are lines in X parallel to Y. Note that the points along any one such line will satisfy the equivalence relation because their difference vectors belong to Y.

Definition 2.12 (Chain Complexes). *Let K be a simplicial complex and p a dimension. A p-chain is a formal sum of p-simplices in K.*

The standard notation for this is $c = \sum a_i \sigma_i$, where the σ_i are the p-simplices and the a_i are the *coefficients*. In computational topology, we mostly work with coefficients a_i that are either 0 or 1, called *modulo 2 coefficients*.

Two p-chains are added componentwise, like polynomials. Specifically, if $c = \sum a_i\sigma_i$ and $c' = \sum b_i\sigma_i$, then $c + c' = \sum(a_i + b_i)\sigma_i$. The p-chains together with the addition operation form the group of p-chains denoted as $(C_p, +)$, or simply, $C_p = C_p(K)$ if the operation is understood.

We have a group of p-chains for each integer p. For p less than zero and greater than the dimension of K this group is trivial, consisting only of the neutral element. To relate these groups, we define the boundary of a p-simplex as the sum of its $(p-1)$-dimensional faces. Writing $\sigma = [u_0, u_1, ..., u_p]$ for the simplex spanned by the listed vertices, its boundary is,

$$\partial_p\sigma = \sum_{j=0}^{p}[u_0, ..., \hat{u}_j, ..., u_p],$$

where the hat indicates that u_j is omitted. For a p-chain, $c = \sum a_i\sigma_i$, the boundary is the sum of the boundaries of its simplices, $\partial_p c = \sum a_i\partial_p\sigma_i$. Hence, taking the boundary maps a p-chain to a $(p-1)$-chain, and write $\partial p : C_p \rightarrow C_{p-1}$. Notice also that taking the boundary commutes with addition, that is, $\partial_p(c + c') = \partial_p c + \partial_p c'$. This is the defining property of a homomorphism, a map between groups that commutes with the group operation. We will therefore refer to ∂_p as the boundary homomorphism or, shorter, the boundary map for chains.

Definition 2.13 (Cycle). *A p-cycle is a p-chain with empty boundary, $\partial_c = 0$.*

Since ∂ commutes with addition, we have a group of p-cycles, denoted as $Z_p = Z_p(K)$, which is a subgroup of the group of p-chains. In other words, the group of p-cycles is the kernel of the $p-th$ boundary homomorphism, $Z_p = \text{ker } \partial_p$

Definition 2.14 (Boundary). *A p-boundary is a p-chain that is the boundary of a $(p+1)$-chain, $c = \partial d$ with $d \in C_{p+1}$*

Since ∂ commutes with addition, we have a group of p-boundaries, denoted by $B_p = B_p(K)$, which is again a subgroup of the p-chains. In other words, the group of p-boundaries is the image of the $(p+1)$-st boundary homomorphism, $B_p = \text{im } \partial_{p+1}$.

Definition 2.15 (Homology). *The $p-th$ homology group is the $p-th$ cycle group modulo the $p-th$ boundary group,*

$$H_p(X) = ker\ \partial_p / im\ \partial_{p+1}$$

Definition 2.16 (P-th Betti Number). *The $p-th$ Betti number is the rank of this group, $\beta_p = rank\ H_p$*

An element of $H_k(X)$ is called homology (equivalence class), and a choice of representative for a class is called a generator. The dimension of $H_k(X)$ counts the number of k-dimensional features of X. For example, dim $H_k(0)$ counts the number of connected components ,dim $H_k(1)$ counts the number of holes, and so on. [4]

2.1.5 Persistent Homology

Persistent Homology can be used to measure the scale or resolution of a topological feature. There are two ingredients, one geometric, defining a function on a topological space, and the other algebraic, turning the function into measurements.

Consider a simplicial complex, K, and a function $f : K \to \mathbb{R}$. Letting m be the number of simplices in K, we get $n + 1 \leq m + 1$ different subcomplexes, which we arrange as an increasing sequence,

$$\emptyset = K_0 \subseteq K_1 \subseteq ... \subseteq K_n = K$$

In other words, if $a_1 < a_2 < ... < a_n$ are the function values of the simplices in K and $a_0 = -\infty$ then $K_i = K = a_i$ for each i. We call this sequence of complexes the *filtration* of $f(x)$, with the radius as x for different simplicial complexes.

For every, $i \leq j$ we have an inclusion map from the underlying space of K_i to that of K_j and therefore an induced homomorphism, $f_p^{i,j} : H_p(K_i) \to H_p(K_j)$, for each dimension p. The filtration thus corresponds to a sequence of homology groups connected by homomorphisms,

$$0 = H_p(K_0) \subseteq H_p(K_1) \subseteq ... \subseteq H_p(K_n) = H_p(K)$$

again, one for each dimension p. As we go from K_{i-1} to K_i, we gain new homology classes, and we lose some when they become trivial or merge with each other. We collect the classes that are born at or before a given threshold and die after another threshold in groups.

Definition 2.17 (Persistent Homology)**.** *The $p - th$ persistent homology groups are the images of the homomorphisms induced by the inclusion, $H_p^{i,j} : im(f_p^{i,j})$ for $0 \leq i \leq j \leq n$.*

Definition 2.18 (P-th persistent Betti Numbers)**.** *The corresponding $p - th$ persistent numbers are the ranks of these groups, $\beta_p^{i,j} = rank H_p^{i,j}$.*

The persistent homology groups consist of the homology classes of K_i that are still alive at K_j or, more formally, $H_p^{i,j} : Z_p(K_i)/(B_p(K_j) \cap Z_p(K_i))$.

We note that births and deaths can also be defined for a sequence of vector spaces that are not necessarily homology groups. All we need is a finite sequence and homomorphisms from left to right which, for vector spaces, are usually referred to as linear maps.

Persistence Diagram. We visualize the collection of persistent Betti numbers by drawing points in two dimensions 2.5. Some of these points may have infinite coordinates and some might be the same, so we really talk about a multiset of points in the extended real plane $\mathbb{R}^2$. Letting $\nu_p^{i,j}$ be the number of p-dimensional classes born at K_i and dying entering K_j, we have

$$\nu_p^{i,j} = (\beta_p^{i,j-1} - \beta_p^{i,j}) - (\beta_p^{i-1,j-1} - \beta_p^{i-1,j}),$$

for all $i < j$ and all p. Indeed, the first difference on the right hand side counts the classes that are born at or before K_i and die entering K_j, while the second difference counts the classes that are born at or before K_{i-1} and die entering K_j. Drawing each point (a_i, a_j) with multiplicity $\nu_p^{i,j}$, we get the $p - th$ persistence diagram of the filtration, denoted as $\text{Dgm}_p(f)$.

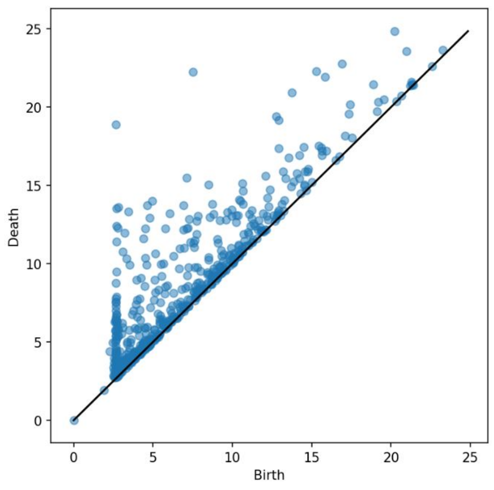


Figure 2.5: Persistence diagram plot extracted from, [17]

2.2 Differentiability of Persistence Diagrams

In this section we are going to understand, how is possible the differentiation of persistence diagrams with respect to its point cloud, [11]. The following explanation will be detailed for Vietoris-Rips filtration. For a more general understanding, I recommend [20].

Let $D_l(X_P)$ be the persistence diagram of a filtration X_P constructed from a finite set P. By choosing the birth and death times in the persistence diagram that are finite, we can express $D_l(X_P)$ as a point

$$v = (b_1, d_1, ..., b_s, d_s, b_{s+1}, ..., b_{s+t}) \in \mathbb{R}^m$$

where $m = 2s + t$. Important to mention that the expression of $D_l(X_P)$ as a point is not unique as the points in the diagonal can permute and give different orders. Then recalling the identification of P and $u \in \mathbb{R}^n$ be a single variable to P, we can consider the persistent homology as giving a single correspondence between the point cloud P and the persistence diagram of a filtration $D_l(X_P)$:

$$\mathbb{R}^n \ni u \to v \in \mathbb{R}^m$$

Now we need to define an appropriate set $O \subset \mathbb{R}^n$ such that this single correspondence is extended to a map $f : O \subset \mathbb{R}^n \to \mathbb{R}^m$ computing persistence diagrams with $f(u) = v$.

Definition 2.19. *A metric space (X, d_x) consists of a non-empty set X and a function $d : X * X \rightarrow [0, \infty)$ such that:*

- *(Positivity) For all $x, y \in x, d_x(x, y) \leq 0$ with equality if and only if $x = y$.*
- *(Symmetry) For all $x, y \in X, d_x(x, y) = d_x(y, x)$.*
- *(Triangle Inequality) For all $x, y, z \in X$*

$$d(x, y) \leq d(x, z) + d(z, y)$$

A function d_x satisfying all conditions, is called a metric on X. [25]

Definition 2.20. *For a metric space (X, d_x) the Hausdorff distance d_H between two subsets $A, B \subset X$ is defined by*

$$d_H(A, B) = maxsup_{a \in A} d_X(a, B), sup_{b \in B} d_X(b, A)$$

where $d_X(a, B) = inf_{b \in B} d_X(a, b)$ and $d_X(b, A)$ is defined symmetrically.

Definition 2.21. *The Gromov-Hausdorff distance d_{GH} between two metric spaces (X, d_Y) and (Y, d_Y) is defined by:*

$$d_{GH}(X, Y) = inf_{f,g} d_H(f_{X \to Z}(X), g_{Y \to Z}(Y))$$

where $f_{X \to Z}$ and $g_{Y \to Z}$ denote isometric embeddings of X and Y into a metric space (Z, d_Z), respectively, and the Hausdorff distance between $f_{X \to Z}(X)$ and $g_{Y \to Z}(Y)$ is measured using the metric d_Z.

Let Δ be the points on the diagonal with multiplicity ∞. The norm $\|p\|_\infty = d_\infty(p, 0)$ by the distance $d_\infty(p, q) =$ max $|p_1 - q_1|, |p_2 - q_2|$ on $\mathbb{R}^2$.

Let P and P' be two point clouds in $\mathbb{R}^L$ with $|P| = |P'| = M$. Then $\delta = d_\infty(D_l(X_P), \Delta)$ and for $\epsilon < \delta/2$, we can let for Vietoris-Rips filtrations,

$$O_u^\epsilon = u' \in \mathbb{R}^n | d_{GH}(P(u), P'(U')) < \epsilon$$

Let $X = (X^r)_{r \in \mathbb{R}_{\leq 0}}$ be a Vietoris-Rips filtration with a saturation time R, and let us set $W_l = X_l^R$ and $W = \cup_l W_l$. The correspondence is decomposed into two parts:

$$u \xrightarrow{\quad g \quad} r = (r_\sigma)_{\sigma \in W} \xrightarrow{\quad h \quad} v$$

Where g constructs the simplicial complex filtration, and h computes the persistence diagram. We extend the decomposition $f = h \circ g$ of this single correspondence to that of a map f. To this aim, we need to construct a proper subset in O_u in which the set W is invariant.

In the case of Vietoris-Rips filtration, W is given by all the faces of the $(M-1)$-simplex for $|P| = M$, independently of the configuration u. Thus, the decomposition $f = h \circ g$ of the correspondence is extended naturally to the map

$$\begin{tikzcd} O_{VR} \arrow[r, "g"] & \mathbb{R}^d \arrow[r, "h"] & \mathbb{R}^m \end{tikzcd}$$

where $O_{VR} = O_u^\epsilon \cap U_{VR}$, $g : O_{VR} \ni u \to r = (r_\sigma)_{\sigma \in W} \in \mathbb{R}^d$, and $h : \mathbb{R}^d \ni r \to v \in \mathbb{R}^m$ with $d = |W|$.

Lemma 2.22. *On the open set O_{VR}, the map g is of class C^∞*

Lemma 2.23. *The map h is of class C^∞ on $\mathbb{R}^d$.*

It can be proved that the map f is of class C^∞ on O_{VR}. This follows from the chain rule and Lemmas 2.22, 2.23.

2.3 Topological Descriptors

A topological descriptor can be understood as a function that expresses the properties of a persistence diagram. Therefore, its topological structure can be understood. This descriptor is what will give us the semantics of a certain point cloud.

For this project, two of them will be defined and will be used for the experimentation of the framework: total persistence and group persistence.

Total Persistence The Total Persistence descriptor is a function that computes the distance of the points with respect to the diagonal. The function is the following:

$$\text{total\_persistence (dgm)} = \sum_{(b_i,d_i)\in \text{dgm}} d_i - b_i$$

This descriptor, used to minimize the total persistence, will make the cloud features vanish, as it will try to approximate the death to little margin from the birth, bringing it closer to the diagonal. For example, in homology 0 it will create connected components from the cloud points. In case of minimizing, $-$total_persistence(dgm), the points will be moved away from the diagonal, in other words, move the death away from the birth. One drawback of this function is that it will not necessarily optimize all points from their respective position. As the descriptor, although it computes the sum of all the points, it does not compute the distance between them or a joint description. For this reason, not all points will be optimized and only some of them may be optimized.

For the training experiment it will be used to maximize the total persistence. This is because it will be possible to better appreciate the optimization in the persistence diagrams. If the total persistence is optimized by minimizing it, there may be problems during the optimization, due to the attempt to make the death smaller than the birth. This is by definition not possible. On the other hand, a positive optimization will be run on a point cloud with many points in order to be able to see how the structure of the point cloud changes during the iterations.

The explanation of the differentiation follows:

Let $f(x)$ be total_persistence(dgm) with a domain of $[0, +\infty)$, because death can't come before birth. To prove that $f(x)$ is differentiable we will use the following theorem: if f is continuous in an open set U and has continuous partial derivatives in U, then f is continuously differentiable at all points in U. Taking $[0, +\infty)$ as the open set we have to see if the partial derivatives are continuous.

$$\frac{\partial f}{\partial b_i} = 1 \qquad \text{and} \qquad \frac{\partial f}{\partial d_i} = -1$$

In both cases, we can see that the partial derivatives are continuous in $(0, +\infty)$. Therefore, the function $f(x)$ is differentiable.

The reason for using this descriptor is because it is a standard in topological data analysis.

Group Persistence The Group Persistence is an own descriptor used to get a relation between the distance between points and total persistence. The function is the following:

$$\text{group\_persistence (dgm)} = f(\text{dgm})\frac{1}{\|(d-b) - \text{mean(dgm)}\|_2 + 1}, \text{ where } d, b \in \text{dgm}$$

The novelty that incorporates the global descriptor is the calculation of the norm with respect to the mean. The mean(dgm) is a function that tries to obtain the mean value between all the points of a persistence diagram, mean(dgm) $= \frac{1}{n}\sum_{i=1}^{n}(\frac{b_i+d_i}{2})$. The subtraction between the vector values and the mean is performed with a previous product of the mean with $[1, ..., 1]$, where the dimension is N, the number of points in the diagram. For example, if $(b - d) = [1, 1, 1]$ and mean= 0.5 the result will be $[0.5, 0.5, 0.5]$. As we will see in the results section, the total persistence in most of the cases increases the distance between a point and the rest of them. The objective is to prioritize that if the points of the diagram are closer together, the result should be closer to total persistence value as the other part should approximate 1.

For the training, the group persistence will be used to maximize the result, as in total persistence. The reasons of using it are to appreciate in which conditions will the network stop to optimize and how far the points will be optimized based on the distance between them.

The explanation of the differentiability follows:

Knowing that f is differentiable, we only have to worry about the other part, due to the fact that the product of differentiable function is also differentiable. When having a division, we will have to see if the function makes a change in the growth, having a value less than 1 in the denominator. The norm between $(d - b) -$ mean(dgm) in the domain of $(0, \infty]$ cannot be less than 0. Let $d, b = [0, .., 0]$, which is the minimum value possible in the domain, because births in homology 0 can only be given at 0. Then we will have a mean value of 0. Thus, the distance between vectors of $[0,..,0]$ will be 0. The minimum of the function will be $\frac{1}{0+1} = 1$. Another possible case for reaching the minimum is when all the points in the diagram are in the same death position, in homology 0. In that case, we take as example $b = [0,0]$ and $d = [1,1]$ and the mean will be 1. When performing $([1,1] - [0,0]) - 1 = [1,1] - 1 = [0,0]$. Therefore, the norm of $[0,0]$ will be 0, leading to the same result as before, $\frac{1}{0+1} = 1$. The addition of the $+1$ is with the purpose of avoiding the division by 0.

Defining $g(x)$ as the group_persistence(dgm) function, with domain in $[0,+\infty)$. Taking the open set $U = [0,+\infty)$, it is necessary to check if the norm in the denominator is differentiable with a finite number of points n in the persistence diagram. We can express $\|(d - b) - \text{mean(dgm)}\|_2 = |(d_1 - b_1 - \text{mean(dgm)})^2|^{1/2} + ... + |(d_n - b_n - \text{mean(dgm)})^2|^{1/2}$, intuitively speaking it is differentiable everywhere except at the origin and all points are death at the same time as the mean. Taking the above expression, one can see that all of them are a composition of two functions, the exponent and absolute value. A proven result is that the absolute value is differentiable everywhere except at the origin. As we are having $\frac{1}{f}$ of a differentiable function, we can use this result [27] to assure that we have differentiability in all the domain except at the origin and all points in the same death as the mean value.

For this particular problem, it would be strange if not impossible to say that all the points lie on the origin axis of the diagram. This would imply that all points in the point cloud to be optimized would be in the same position. Also, in terms of experimentation it will be interesting to see in which circumstances the death points will match the mean value.

2.4 Deep Learning

So many times Deep Learning has been heard that it is a revolution in Artificial Intelligence. But what exactly is Deep Learning? To understand it better, lets take a look at the history.

In 1943, artificial neural networks (ANNs) were an attempt to mimic biological neurons in the human brain by neurophysiologist Warren McCulloch and mathematician Walter Pitts. This work was set forth in "A Logical Calculus of Ideas Immanent in Nervous Activity," McCulloch and Pitts presented a simplified computational model of how biological neurons could work together in animal brains perform complex computations using propositional logic. It was the first artificial neural network architecture [12].

Now to begin building what is called Artificial Neural Network, let’s start by defining it. Following that, going to define Multi Layer Perceptron’s the Deep Neural Network used in this project and how to optimize it.

2.4.1 Artificial Neural Network

This explanation is mainly extracted from [16].

An Artificial Neural Network consists of a network of simple information processing units, called neurons. The power of neural networks to model complex relationships is not the result of complex mathematical models, but rather emerges from the interactions between a large set of simple neurons.

A neuron2.6 receives n inputs $[x_1, .., x_n]$ on n different inputs connections, and each connection has an associated weight $[w_1, .., w_n]$. The weighted sum calculation involves the multiplication of inputs by weights and the summation of the resulting values.

$$z = (x_1 \cdot w_1 + (x_2 \cdot w_2) + .. + (x_n \cdot w_n)) = x \cdot w$$

Example 2.24. Assuming a neuron received the inputs $[x_1 = 3, x_2 = 9]$ and had the following weights $[w_1 = -3, w_2 = 1]$, then $z = (3 \cdot -3) + (9 \cdot 1) = 0$

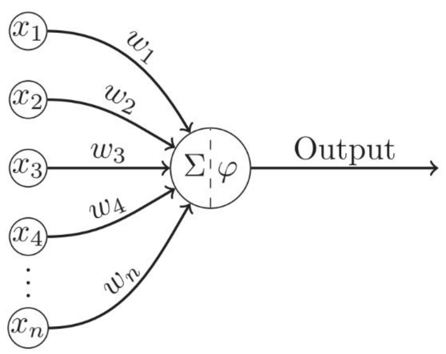


Figure 2.6: The structure of an artificial neural network, [16]

After getting the output of a neuron, an activation function has to be associated with it. The reason for using the function in the output of the neuron value z is to switch from a linear to a nonlinear mapping. A linear model can always be represented by a neuron, so large networks do not help if nonlinear activations are not available. Also, the need to obtain a nonlinear mapping is because many relationships in the world that may be modelled are nonlinear.

The fact that the introduction of a nonlinearity into the processing of the neurons enables the network to learn. A nonlinear mapping between inputs and output is another illustration of the fact, that the overall behaviour of the network emerges from the interactions of the processing carried out, by individual neurons within the network. Neural networks solve problems using a divide-and-conquer strategy: each of the neurons in a network solves one component of the larger problem, and the overall problem is solved by combining these component solutions.

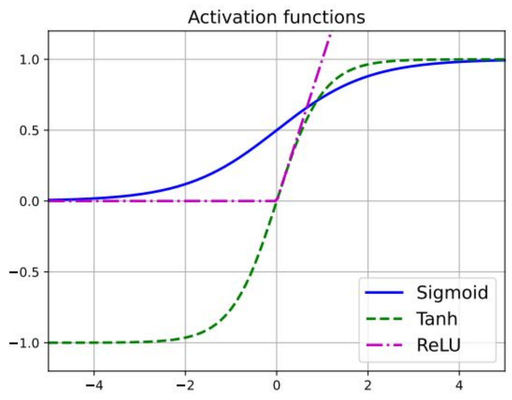


Figure 2.7: Plots of some neural network activation functions, [24]

In this project, the ReLU function, which has the following notation, will be used:

$$f(x) = max(0, x)$$

The reason for choosing ReLU function was because of its computational efficiency, and it converges faster to optimal solutions than the rest due to its sparsity.[1]

In order to adjust the weights of the neuron, its error must be calculated and depending on this the weights can be adjusted so that they are more similar to the z that we want to associate the x inputs. To do this a loss function must be defined, which has the following definition.

Definition 2.25 (Loss function)**.** *The loss function is a function that calculates the error of the actual value with respect a prediction. Importantly, this function must be differentiable in order to optimize the error.*

The optimization of a neuron can be done with the last two sections of this section, as well as the Multi Layer Perceptron.

2.4.2 Multi Layer Perceptron

This section is mostly extracted from [13].

Deep feedforward networks, also called feedforward neural networks, or multi layer perceptrons (MLPs), are the quintessential deep learning models.The goal of a feedforward network, and all neural networks, is to approximate some function f^*.

Example 2.26. For a class y, $y = f^*(x)$ maps an input x to a category y. A Feedforward network defines a mapping $y = f(x; \theta)$ and learns the value of the parameters θ, weights, that result in the best function approximation. According to the values that minimizes the loss function.

Those models are called feedforward because information flows through the function being evaluated from x, through the intermediate computations used to define f, and finally to the output y. No feedback connections exist where the outputs of the model feed back to themselves.

Feedforward neural networks are typically represented by composing together many different functions. The model is associated with a directed acyclic graph describing how the functions are composed together.

The following example shows how to realize a simple MLP architecture. First, the meaning of a layer will be defined.

Definition 2.27 (Layer)**.** *An artificial neural network (1) takes some input data, and (2) transforms this input data by calculating a weighted sum over the inputs and (3) applies a non-linear function to this transformation to calculate an intermediate state. The three steps above constitute what is known as a layer, and the transformative function is often referred to as a unit. A layer in a deep learning model is a structure or network topology in the architecture of the model, which is a container that usually receives weighted input, transforms it with a set of mostly non-linear functions and then passes these values as output to the next layer.[9]*

There are three types in a neural network:

- **Input layer**: Contains the data to process: images, text or any kind of data represented in number.
- **Hidden Layer**: All the layers that can be found between the input and output layer, are considered hidden. They try understand the problem and fit themselves to approximate to f^*.
- **Output Layer**: The different number of classes in the classification assigns a score to the input image, text, or the data introduced to the belonged class. In case of a regression problem a single output neuron will give us the scalar computed by the network.

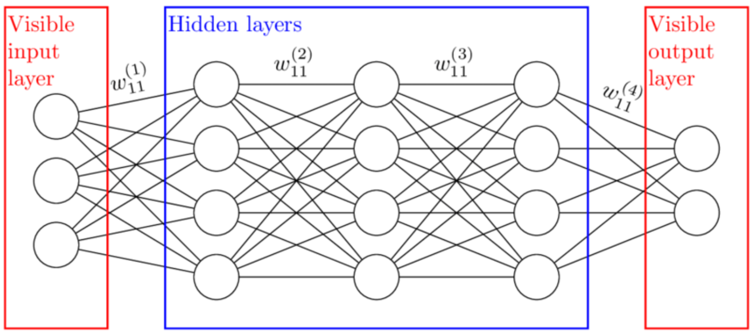


Figure 2.8: An example of a deep learning neural network with 3 hidden layers., [2]

Now let's take an example of what a Feedforward neural network would look like:

Example 2.28. There are three functions f^1, f^2, and f^3 connected in a chain, to form $f(x) = f^3(f^2(f^1(x)))$. In this case f^1 is the first layer, f^2 is the second and so on.

The overall length of the chain gives the depth of the model. The name "deep learning" arose from this terminology.

2.4.3 Backpropagation

This section is mostly extracted from [24].

The backpropagation algorithm can be used to compute the gradient of a loss function applied to the output of the network with respect to the parameters in each layer. This gradient can then be passed to a gradient-based optimization algorithm.

Let assume initially that the neural network is a simple linear chain of stacked layers, as in an MLP. In this case, backpropagation is equivalent to repeated applications of the chain rule of the calculus. However, the method can be generalized to arbitrary directed acyclic graphs. Such a general procedure is often referred to as automatic differentiation or autodiff.

To better understand the concepts of the backpropagation algorithm, let's give as an example that the network has 4 layers, 2.9.

Consider a mapping of the form $o = f(x)$, where $x \in \mathbb{R}^n$ and $o \in \mathbb{R}^m$. Assume that f is defined as a composition of functions: $f = f_4 \circ f_3 \circ f_2 \circ f_1$, where $f_1 : \mathbb{R}^n \to \mathbb{R}^{m_1}$, $f_2 : \mathbb{R}^{m_1} \to \mathbb{R}^{m_2}$, $f_3 : \mathbb{R}^{m_2} \to \mathbb{R}^{m_3}$ and $f_4 : \mathbb{R}^{m_3} \to \mathbb{R}^{m_4}$.

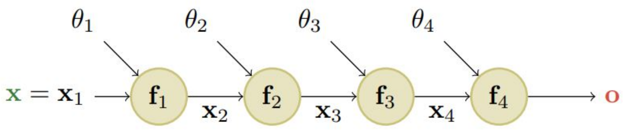


Figure 2.9: A simple linear-chain feedforward model with 4 layers. Here x is the input and o is the output., [24]

The Jacobian can be calculated $J_f(x) = \frac{\partial o}{\partial x^T} \in \mathbb{R}^{mxn}$ using the chain of rule:

$$\frac{\partial o}{\partial x} = \frac{\partial o}{\partial x_4}\frac{\partial x_4}{\partial x_3}\frac{\partial x_3}{\partial x_2}\frac{\partial x_2}{\partial x} = \frac{\partial f_4(x_4)}{\partial x_4}\frac{\partial f_3(x_3)}{\partial x_3}\frac{\partial f_2(x_4)}{\partial x_2}\frac{\partial f_1(x)}{\partial x} = J_{f_4}(x_4)J_{f_3}(x_3)J_{f_2}(x_2)J_{f_1}(x)$$

Recall that

$$J_f(x) = \frac{\partial f(x)}{\partial x} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{pmatrix} = \begin{pmatrix} \nabla f_1(x)^T \\ \vdots \\ \nabla f_m(x)^T \end{pmatrix}$$

where $\nabla f_i(x)^T \in \mathbb{R}^{1xn}$ is the i-th row and $\frac{\partial f}{\partial x_j} \in \mathbb{R}^m$ is the j-th column.

Using a vector Jacobian product, VJP of the form $e_i^T J_f(x)$, where $e_i^{R^m}$ is the unit basis vector, can be used to extract the i-row of $J_f(x)$. Similarly, one can extract the j-th column of $F_f(x)$ using a Jacobian vector product, JVP, of the form $J_f(x)e_j$, in which $e_j \in \mathbb{R}^n$. This shows that the computation of $J_f(x)$ reduces to either n JVPs or m VJPs.

If $n < m$, it is more efficient to compute $J_f(x)$ for each column $j = 1 : n$ by using JVPs in a right-to-left manner. The right multiplication with a column vector v is:

$$J_f(x)v = J_{f_4}(x_4) \cdot J_{f_3}(x_3) \cdot J_{f_2}(x_2) \cdot J_{f_1}(x_1) \cdot v$$

This can be done using forward mode differentiation 7. Assuming $m = 1$ and $n = m_1 = m_2 = m_3$ the cost of computing $J_f(x)$ is $O(n^3)$

Algorithm 1 Reverse mode differentiation

```
$x_1 := x$
$v_j := e_j \in \mathbb{R}^n forj = 1 : n$
**for** $k = 1 : K$ **do**
    $x_{k+1} = f_k(x_k)$
    $v_j = J_{f_k}(x_k)v_j forj = 1 : n$
**end for**
Return $o = x_{K+1}, [J_f(x)]_{:,j} = u_i^T$ for $j = 1 : n$
```

If $n > m$, e.g. if the output is a scalar, it is more efficient to compute $J_f(x)$ for each row $i = 1 : m$ using VJPs in a left-to-right manner. The left multiplication with a row vector u^T is:

$$u^T J_f(x) = u^T \cdot J_{f_4}(x_4) \cdot J_{f_3}(x_3) \cdot J_{f_2}(x_2) \cdot J_{f_1}(x_1)$$

This can be done using reverse mode differentiation 9. Assuming $m = 1$ and $n = m_1 = m_2 = m_3$ the cost of computing $J_f(x)$ is $O(n^2)$

Algorithm 2 Reverse mode differentiation

```
$x_1 := x$
**for** $k = 1 : K$ **do**
    $x_{k+1} = f_k(x_k)$
**end for**
$u_i := e_i \in \mathbb{R}^m$ for $i = 1 : m$
**for** $k = K : 1$ **do**
    $u_i^T := u_i^T J_{f_k}(x_k)$ for $i = 1 : m$
**end for**
Return $o = x_{K+1}, [J_f(x)]_{i,:} = u_i^T$ for $i = 1 : m$
```

Now each layer can have parameters $\theta_1, ..., \theta_4$ in an MLP. Our focus is on the case a mapping has the form $L : \mathbb{R}^n \to \mathbb{R}$, so the output is a scalar.

The notation $f_k(x_k, \theta_k)$ denotes the function at layer k, where x_k is the previous output and θ_k are the optional parameters for this layer.

The algorithm 3 computes the gradient of the loss with respect to the parameters in each layer. Also calculates the gradient of the loss with respect to the input, $\nabla_x L \in \mathbb{R}^n$, in which n is the dimensionality of the input.

Algorithm 3 Backpropagation for an MLP with K layers

```
// Forward pass
$x_1 := x$
for $k = 1 : K$ do
    $x_{k+1} = f_k(x_k, \theta_k)$
end for
// Backward pass
$u_{K+1} := 1$
for $k = K : 1$ do
    $g_k := u_{k+1}^T \frac{\partial f_k(x_k,\theta_k)}{\partial \theta_k}$
    $u_k^T := u_{k+1}^T \frac{\partial f_k(x_k,\theta_k)}{\partial x_k}$
end for
Return $L = x_{k+1}, \nabla_x L = u_1, \nabla_{\theta_k} L = g_k : k = 1 : K$
```

To know how to compute the vector Jacobian product (VJP) of all supported layers can be found in [24] in section 13.3.3.

2.4.4 Optimization

[24]The core problem of machine learning is parameter estimation. This requires solving an optimization problem, where to find the values for a set of variables $\theta \in \Theta$, minimizing a scalar-valued loss function $L : \Theta \to \mathbb{R}$:

$$\theta^* \in \arg\min_{\theta \in \Theta} L(\theta)$$

The parameter space will be assumed to be given by $\Theta \subseteq \mathbb{R}^D$, where D is the number of variables being optimized over. Therefore, the focus is on continuous optimization, rather than discrete optimization.

To achieve the optimization of a network's parameter, the gradient descent algorithm is the solution. In order to compute it first, it is necessary to start with some random weight values, the first ones to be assigned to the network. Once obtained, at each iteration t, we perform the following update for each of them:

$$\theta_{t+1} = \theta_t + n_t d_t$$

where n_t is known as the step size or learning rate, and d_t is a descendent direction, such as the negative of the gradient. These update steps are continued until the method reaches a stationary point, where the gradient is zero.

A direction d is said to be a decreasing direction if there is a sufficiently small amount d with which to move in the direction d and the value of the function is guaranteed to decrease. How to obtain this direction from a neural network is explained in the section on backpropagation.

In machine learning, the sequence of step sizes$\{n_t\}$ is called the learning rate schedule. There are several widely used methods for picking this. The two main ones are explained:

- **Constant step size**: The simplest method is tu use a constant step size. However, if it is too large, the method may fail to converge, and if it is too small, the method will converge very slowly.
- **Line search**: Te optimal step size can be found by finding the value that maximally decreases the objective along the chosen direction by solving the 1d minimization problem:
  $$n_t = \arg\min_{n>0} L(\theta_t + nd_t)$$
  However, it is not usually necessary to be so precise.

Gradient descent can move very slow along flat regions of the loss landscape. One simple heuristic, known as the momentum, is to move faster along directions that were previously good, and to slow down along directions where the gradient has suddenly changed.

Adaptive Moment Estimation (ADAM), which incorporates this momentum feature, will be the optimizer used for this project. It is currently used as a standard in network training.

Chapter 3

Experiments

At this point, all the theory required to develop the project has been covered. First, a mention must be made of an earlier attempt by Mathieu Carrière, [5] where he initiated the optimization of simple point clouds on a Jupyter Notebook. The experiments to test the functionality of this software and how it was done will be explained.

3.1 Framework Pipeline

Now the data flow has to be designed to achieve the optimization, based on Topological Descriptors. As mentioned above, it is important to emphasize the use of the ReLU function because of its computational efficiency in respect to the other nonlinear functions.

After having the activation vectors, the Persistence Diagram with the Euclidean distance between the vectors needs to be extracted, this will be explained in more detail in 3.6. Once the diagram is obtained, it is necessary to define the topological descriptor to optimize and calculate the derivative with the TensorFlow auto-differentiation Tape.

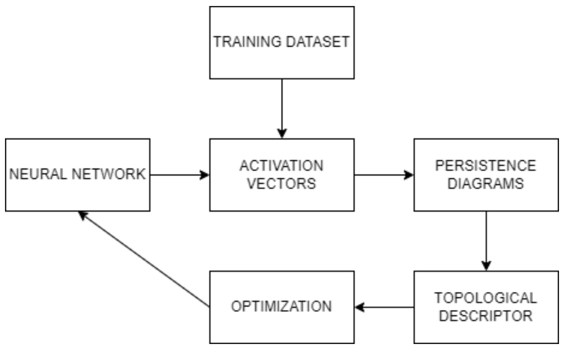


Figure 3.1: Pipeline constructed for the framework

3.2 Implementation Details

In this section, the components used to work with this project and the main Python libraries used will be shown. In addition, there is a requirement text file in the code to run the code by yourselves.

Hardware Components:

- **CPU**: Intel(R) Core(TM) i5-10210U CPU
- **GPU**: NVIDIA GeForce MX250
- **RAM**: 8.00 GB
- **Operative System**: Windows 10
- **Kind of System**: Operative system of 64 bits.

I have to mention that given the limitation with these resources, I did not test the process with all the datasets. I created a function called *reduce_dataset*, which take for arguments the images, labels, and the percentage of reduction. On my case, only 0.01 of the randomly selected training data was taken.

Python libraries:

- **TensorFlow**: TensorFlow is an interface for expressing machine learning algorithms and an implementation for executing those algorithms [23]. TensorFlow can be used to define Deep Neural Networks of any type, CNNs, MLPs, RNNs, etc. One of the major advantages is the self-differentiation, which along with the Tape object records the operation performed in a process and optimizes with the recorded operations.
- **GUDHI**:The Gudhi library is an open source library for Computational Topology and Topological Data Analysis (TDA). It offers state-of-the-art algorithms to construct various types of simplicial complexes, data structures to represent them, and algorithms to compute geometric approximations of shapes and persistent homology [22]. This library is used in the project to calculate the persistence diagrams given the activation vectors.
- **NumPy**: NumPy's main object is the homogeneous multidimensional array. It is a table of elements (usually numbers), all the same type, indexed by a tuple of non-negative integers [14]. The purpose of the project is to cast from TensorFlow tensors into arrays to make the storage easier and the reading of the data.

Some other Python libraries such as pickle, os, matplotlib, pandas have also been used. Since these libraries were used during the undergraduate course and the use of the framework was not deterministic, these do not need to be explained in detail.

3.3 Datasets

To analyze the optimization of topological descriptors, it was decided to use 3 datasets of different complexity in a MLP. Regarding the training data used, it was only needed for the training step, because the only part to test was that the optimization worked given the inputs in the network.

The datasets used were:

- **CIFAR10** [18]: The problem which tries to solve is a classification problem of computer vision. In this problem, we have 10 different classes, which can be found in the TensorFlow documentation. It contains 60000 RGB images with dimension $(32, 32, 3)$, for the training has 5000 giving a tensor of $(6000, 32, 32, 3)$.
- **CIFAR100** [18]: This dataset is very similar with respect to CIFAR10, but the complexity of the problem is bigger as a result of it. It has 100 different classes instead of 10, containing 600 images per the 100 classes, and 500 of them are used for training. Also, it has the same shape image as CIFAR10, $(32, 32, 3)$.
- **MNIST**[19]: The MNIST dataset is a collection of handwritten digits. It has a training of 60000 non-RGB images with a dimension per image of (28, 28). This is also a problem of classification in computer vision, which tries the handwritten digit tries to be recognized by the computer.

3.4 Deep Learning architectures

As the experiment wants to be able to make a general framework, 4 different architectures were used to see if the experiment works freely on the chosen architecture.

The number of layers for each model was: 2, 3, 5, 10. Which were decided to study if a network with more layers was easier to optimize its topological structure. All layers were 100 neurons, the number of neurons was to place a standard and with the hardware resources it was possible to train the network. Finally, the activation function is the ReLU as mentioned in the theory.

3.5 From Graph to a Space Representation

During this section, for the construction of the activation vector, this algorithm will be explained and how it works.

In this project, the ReLU function of neurons defined on artificial neural networks is used. This transformation is based on the fact that a network behaves dynamically only in the context of the input data, forming a matrix of neuron activations.

To compute the activation vectors, a D dataset and a m model are needed. Then the first layer has to be skipped, since it has no impact on the outcome of the network. Then, the construction of the *activation_bd* starts by obtaining the values of the neurons in each layer. Once the activations are obtained, it starts concatenating the columns of the neuron activations in the matrix.

Before returning the *activation_bd*, it is necessary to execute the transpose operation at the matrix. This is in order to get each neuron as a row and each sample of data as a column.

Algorithm 4 From Neural Network to Activation_vector

```
// The arguments for the code are model MLP and input train
first_layer = True
for $layer$ in $model.layers$ do
    if $layer$ is $flatten\_layer$ then
        Skip
    else
        activation_vector = layer(input)
        if first_layer then
            activations_bd = activation_vector
            first_layer = False
        else
            activation_bd = concat([activations_bd, activation_vector], axis = 1)
        end if
    end if
end for
Return activations_bd transposed
```

At the end, a matrix will be obtained, it can be understood in the following way: each row indicates the position of that neuron in the activation with respect to the data entered in the network. The topological optimization will be based on this structure. An important clarification is that labels are not needed when it comes to the performance of this matrix.

3.6 Persistence Diagram Computation

The computation of the persistence diagrams will now be detailed. Indeed, most of the information in this part is taken from Mathieu Carrière, whose original repository can be found in the bibliography. First of all, to connect this part with all that has been previously explained, the activation vectors have already been calculated. This has been explained in the theory section 2.5.

To begin with, the matrix of distances between the different vectors must be calculated. This matrix will be of size NxN, where N is the number of neurons, the length of this being the number of data samples used to calculate the activation vector. The Euclidean distance has been used to achieve the results presented, but the code is supported to use either with TensorFlow operations. As other distances are not detailed in the theory, we will only focus on the Euclidean distance in this project.

Once the distance matrix is obtained, the Rips function is executed. It is made with the Gudhi library and NumPy, being required the dimension in which the homology is to be performed, also it is needed the maximum distance for the filtration and the number of points of the diagram to collect. The latter is due to the fact that on the diagonal there are infinite points, and it is to be able to compute in a limited way.

As it is executed, the `RipsTF` function, once executed, will return the indices in which the homology persistence dies in regard to the distance matrix. At the end, some transformations of the births and deaths are performed in order to obtain the 2-dimensional persistence diagram, with the previously specified number of points.

This is how the persistence diagram is obtained. Now, the optimization itself has been implemented in a more technical way.

3.7 Optimization of activation vectors

All that has been explained and shown previously, it can be put together in order to achieve the objective of the project.

The most important thing is to execute inside tf.GradientTape() the operations of calculating the activation vectors, the persistence diagram and the topological descriptor. Once the topological descriptor is obtained, it is needed to be taken as the loss value to be optimized. Then compute the gradients with the tape, remember that the tape saves the operations previously performed. Once the gradient is obtained, we optimize with ADAM the Multi Layer Perceptron and this is how the network is optimized in an epoch with its topological complexity.

In order to optimize the model with this pipeline, it is important to keep the batch static in the epochs. This is because otherwise the activation vectors will change and optimize according to different values.

Chapter 4

Results

In order to determine how successful the framework was, two different types of tests were run. The first test was developed for checking if a two-dimensioned point cloud optimizes its structure using homology in dimension 0. The second test consisted of running the aforementioned pipeline through Neural Network basic procedures. The generated results and their analysis will now be explained in more detail.

4.1 Point Cloud Optimization

Once the topological optimizer, which takes the point cloud's persistence diagram as a reference, was completed, it was decided to run the following test for checking its good performance. This validation test consists of visualizing a 2-dimensional point cloud generated randomly using NumPy. For the purpose of verifying its performance, 300 epochs have been run, seeking the optimization of the total persistence. This aims to reduce it.

By using homology's dimension 0, as discussed in the theory section, the mentioned test has to evaluate the connected components within the cloud. Then, if our goal is to optimize this dimension 0, newly generated cloud components should appear as a result of this test.

As can be seen in 4.1, the optimization of the Point Cloud can be clearly observed. For instance, at epoch 0, a point cloud has been randomly generated, where no clear structure is visible. From the 100-th epoch onwards, the first components to be connected can be seen, in which points are joined together for building them. Moving forward to the 200-th epoch, the sought optimization can be appreciated in a clearer way, since the isolated points start to get coupled with the already constructed components. Finally, by the 300-th iteration, not much change can be seen. This is caused by the fact that, from the 200-th epoch, the total persistence function has already generated components with the points.

From this result, it was possible to ensure the good performance of the Framework in the task of optimizing the point clouds using the homology in dimension 0. After this point on, it was possible to continue the work for performing this optimization with the activation vectors of the neural networks.

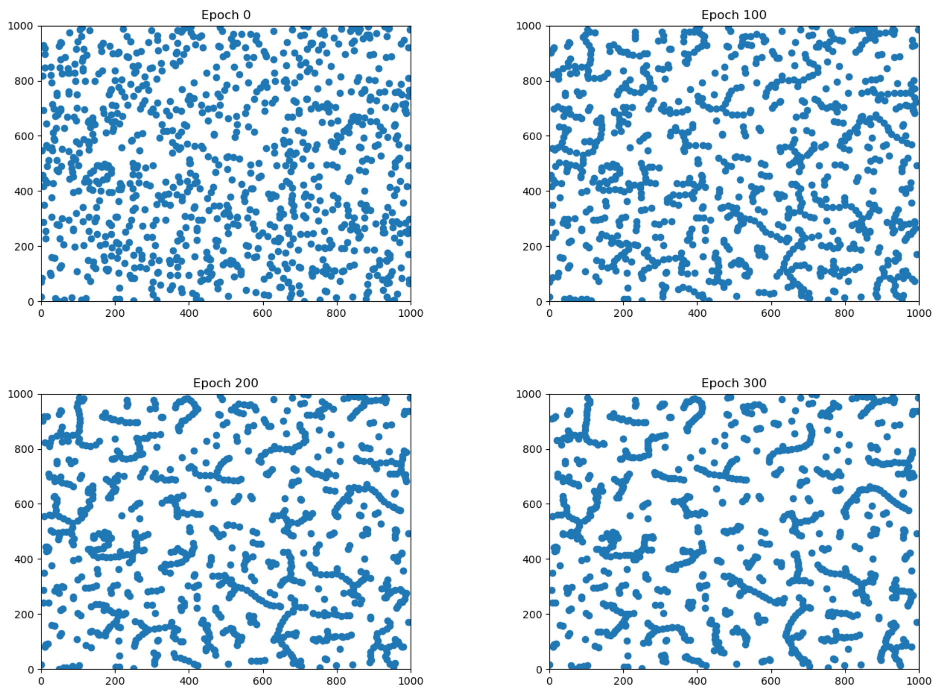


Table 4.1: Optimization of 300 iterations on a point cloud, achieved with homology dimension 0 and total persistence as a loss function

4.2 Multi Layer Perceptron Topology Optimization

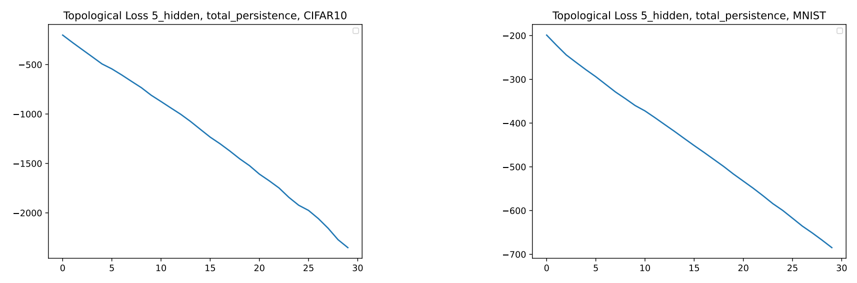
This section will aim to show the performance of topological optimization in neural networks. The results for two different descriptors will be analyzed for evaluating whether they meet the desired requirements or not. Both experiments were studied within 30 iterations. It should be noted that the same default hyper-parameters have been used so that the final result is not affected by them. Since dimension 0 is applied for the persistent homology of Vietoris-Rips, the results are presented using density functions. The reason for this is that, in the persistence diagrams, $(x, y) \in \mathbb{R}^2$, the X-axis will always be 0. In other words, the birth will always occur at 0 and its death will be optimized.

4.2.1 Total Persistence Results

In 1,3,5, one may observe density functions, generated in the activation vectors of the first epoch, the non-optimized network and the last optimized network. As with both CIFAR10 and CIFAR100, regardless of the architecture, one feature always stands out, dramatically increasing its overall persistence compared to the rest. However, when comparing with MNIST, the features seem to be more clustered than with the other two datasets. Especially the 5 hidden layer architecture seems to fulfill what is intended by the grouped persistence.

Taking a look at the values of the loss function 4.1, total persistence reduces the loss in a linear way. This is because, when trying to maximize the total persistence, the death of these features should last longer in time. From the persistence diagrams and the density functions, in all the epochs, it can be appreciated that these characteristics last longer in death.

Although the objective of the optimization is met, it would be interesting to find a way to achieve a more generic optimization among all the points. Instead of just optimizing one point, we should try to optimize all of them. That is why now it is time to look at the optimization of the group persistence descriptor and see the results it gives us.



(a) Loss in CIFAR10 with 5 hidden layers (b) Loss in MNIST with 5 hidden layers

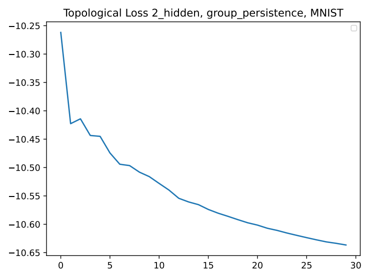
Figure 4.1: Optimization with total persistence during 30 epochs

4.2.2 Group Persistence Results

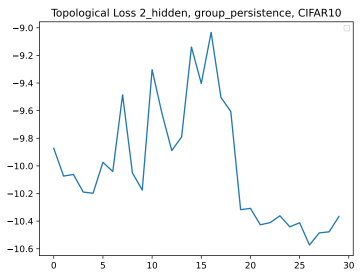
In 2, 4, 6, the results generated with group persistence can be seen. First of all, it can be appreciated that in all the training cases there is no point that is very far away from the rest, as was the case with total persistence. Therefore, the optimization goal with group persistence has been achieved. Furthermore, we can see that the final results offer us information to new hypotheses.

The difficulty within the three datasets differs among them. The order from the easiest to the most difficult classification problems for a network to learn the dataset of is MNIST, CIFAR10 and CIFAR100. In MNIST, the optimization ends up with all the points closely grouped, the mean and the points of death are similar values and the norm tends to 0, therefore the descriptor is no longer differentiable. Being a result quite different from those presented by CIFAR10 and CIFAR100, one might think that it is due to the dataset itself. However, if one looks in detail at the result of the 10 hidden layers architecture in CIFAR10, the points have a very similar behaviour to MNIST. The hypothesis that arises from these facts is the following: *"The network has topological characteristics that will have very similar persistence if the problem to be solved for that network is simple"*. To see if this hypothesis is possible, CIFAR100, the most difficult problem, does not have this behaviour in any of its architectures. Therefore, although the hypothesis cannot be proved due to lack of evidence, it cannot be discarded either.

Finally, the loss function of group persistence is not as stable as the one from total persistence. Unlike CIFAR10 and CIFAR100, the loss function of MNIST, in all its architectures, is constant. In any case, although it is not as stable, we can observe that it ends up optimizing correctly. This is due to the fact that it first prioritizes the total persistence, which causes some points to move away from the rest. But then one can see that it tries to correct the loss value by making the rest of the group go towards the mean. Then it can be seen that during the process it does meet the descriptor objective more firmly, since it increases the total persistence without making one topological feature stand out much more than the others.

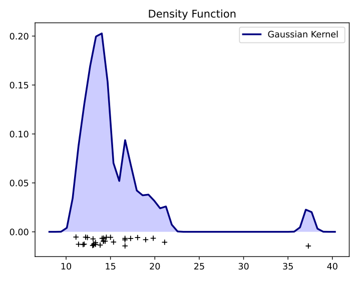


(a) Loss in MNIST with 2 hidden layers

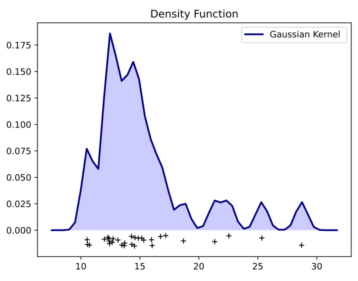


(b) Loss in CIFAR10 with 2 hidden layers

Figure 4.2: Optimization with group persistence during 30 epochs



(a) Density function in epoch number 15



(b) Density function in epoch number 16

Figure 4.3: Density functions in 2 hidden layers of CIFAR10. From epoch 15 to 16, the distance between points are minimized.

Chapter 5

Conclusions

As the explanation of the framework and the results obtained are now over, a summary of the project will follow.

A function that transforms a neural network to a vector space? has been successfully achieved, being able to obtain topological information of the network from persistence diagrams and differentiate them to mold the structure to the descriptor of interest. For the moment we can only optimize with the homology in dimension 0, and it would be interesting to see the behaviour on other dimensions. However, we have ensured a good performance by seeing how it modifies the structure of a point cloud.

The differentiability of persistence diagrams has been tested with two topological descriptors and three different datasets, CIFAR10, CIFAR100 and MNIST, where the difficulty of these classification problems has been different. All this in order to guarantee that the framework optimizes independently from these three datasets. An important point to make, when optimizing with regard to the topological complexity of the network, is that we do not use ground truth. Unsupervised machine learning techniques are gaining popularity due to the contribution of finding new patterns in the data. In this case, seeing new patterns in networks could be a subject of discussion.

The topological optimization on activation vectors for future regularization has been the innovative part of the project. Although there have already been attempts to apply topological regularizers [8, 15, 4] and topological optimization to point clouds [11, 21], there has been no previous attempt to combine these two fields with the work of Ballester et al. [3]. Also the idea of this framework is to explore and test new ways of representing the network and compare the results obtained for the scientific community.

Although the optimization results with total persistence and group persistence work, we see that both have different behaviours. In group persistence, the optimization is affected by the difficulty of the problem and the architecture of the network, minimizing the distance between the points of the persistence diagram. This may be an indication that it may come to function as a regularizer in the optimization so that the network can generalize.

On the other hand, we see that when optimizing with full persistence, a very similar behaviour is obtained regardless of the architecture or the dataset. This may make it more interesting in order to investigate other types of descriptors such as the group descriptor.

In conclusion, we have been able to meet the objectives of optimizing a neural network based on its topological complexity. This has been possible thanks to the understanding of the persistent homology tool and the previously cited works.

5.1 Future Work

There are still many things to do for this project. One possible enhancement would be the framework allowing you to choose new types of simplicial complexes for optimization, as well as working in different homology dimensions. Also, the exploration of new ways to represent the network as a point cloud, and check which one of these representations are the best for these optimizations.

It would also remain to be seen how they optimize other topological descriptors, whether landscapes or entropy in persistence diagrams.

Another type of tests that could be performed for the performance is that of new types of networks such as recursive neural networks or convolutional neural networks. Also test with new types of problems such as sentiment analysis in text or graph problems with graph neural networks.

The final objective and continuation of the project is to try to integrate this topological optimization to the standard training of neural networks, performing a topological regularizer as has been attempted to date to counteract the problem of overfitting. The idea of this integration can also open new lines towards the interpretability of the networks. As already done in the work of Ballester et al. [3] the generalization of networks can be explained from these topological tools. So coupling this way of explaining networks to their training may be interesting as a line of research.

As can be seen, there is still a lot of research and testing to be done in the topological optimization of neural networks. Not only that, but also to try to solve the problem that has been tried to be solved for a long time in machine learning research, overfitting. The combination of the study of topological data analysis tools and deep learning is starting a revolution and this work has tried to make its contribution the field.

Bibliography

[1] RockTheStar (https://stats.stackexchange.com/users/41749/rockthestar). *What are the advantages of ReLU over sigmoid function in deep neural networks?* Cross Validated. URL:https://stats.stackexchange.com/q/126238 (version: 2019-10-24). eprint: https://stats.stackexchange.com/q/126238. URL: https://stats.stackexchange.com/q/126238.

[2] Jeremy Adcock et al. "Advances in quantum machine learning". In: (Dec. 2015).

[3] Rubén Ballester et al. *Towards explaining the generalization gap in neural networks using topological data analysis*. 2022. DOI: 10.48550/ARXIV.2203.12330. URL: https://arxiv.org/abs/2203.12330.

[4] Rickard Brüel-Gabrielsson et al. "A Topology Layer for Machine Learning". In: *Paper* (2020).

[5] Mathieu Carriére. *difftda*. URL: https://github.com/MathieuCarriere/difftda.

[6] Wikipedia contributors. *Group (mathematics)*. URL: https://en.wikipedia.org/wiki/Group_(mathematics).

[7] Wikipedia contributors. *Quotient space (linear algebra)*. URL: https://en.wikipedia.org/wiki/Quotient_space_(linear_algebra).

[8] Chao Chen et al. "A Topological Regularizer for Classifiers via Persistent Homology". In: *Paper* (2018).

[9] Tim Dettmers. *Deep Learning in a Nutshell: Core Concepts*. URL: https://developer.nvidia.com/blog/deep-learning-nutshell-core-concepts/.

[10] Herbert Edelsbrunner and John Harer. *Computational Topology An Introduction*. Duke University, 2010.

[11] Marcio Gameiro, Yasuaki Hiraoka, and Ippei Obayashi. "Continuation of point clouds via persistence diagrams". In: *Physica D: Nonlinear Phenomena* 334 (2016), pp. 118–132. DOI: 10.1016/j.physd.2015.11.011. URL: https://doi.org/10.1016%2Fj.physd.2015.11.011.

[12] A. Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media.

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

[14] Charles R. Harris et al. “Array programming with NumPy”. In: *Nature* 585.7825 (2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[15] Xiaoling Hu et al. “Topology-Preserving Deep Image Segmentation”. In: *Paper* (2019).

[16] John D. Keller. *Deep Learning*. The MIT Press Essential Knowledge Series.

[17] Gary Koplik. *Persistent Homology: A Non-Mathy Introduction with Examples*. URL: https://towardsdatascience.com/persistent-homology-with-examples-1974d4b9c3d0.

[18] Alex Krizhevsky. *Learning multiple layers of features from tiny images*. Tech. rep. 2009.

[19] Yann LeCun, Corinna Cortes, and CJ Burges. “MNIST handwritten digit database”. In: *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist* 2 (2010).

[20] Jacob Leygonie, Steve Oudot, and Ulrike Tillmann. *A Framework for Differential Calculus on Persistence Barcodes*. 2019. DOI: 10.48550/ARXIV.1910.00960. URL: https://arxiv.org/abs/1910.00960.

[21] Jacob Leygonie, Steve Oudot, and Ulrike Tillmann. *A Framework for Differential Calculus on Persistence Barcodes*. 2019. DOI: 10.48550/ARXIV.1910.00960. URL: https://arxiv.org/abs/1910.00960.

[22] Clément Maria et al. “Rips complex”. In: *GUDHI User and Reference Manual*. GUDHI Editorial Board, 2016. URL: http://gudhi.gforge.inria.fr/doc/latest/group__rips__complex.html.

[23] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[24] Kevin P. Murphy. *Probabilistic Machine Learning: An introduction*. MIT Press, 2022. URL: probml.ai.

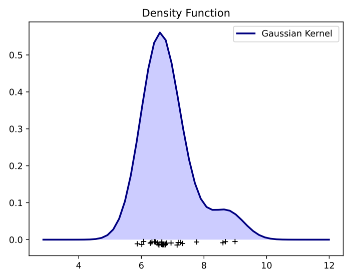
[25] Universidad de Oslo. *Metric.pdf*. URL: https://www.uio.no/studier/emner/matnat/math/MAT2400/v11/Metric.pdf.

[26] Chad M. Topaz, Lori Ziegelmeier, and Tom Halverson. “Topological Data Analysis of Biological Aggregation Models”. In: *PLOS ONE* 10.5 (2015). Ed. by Bard Ermentrout, e0126383. DOI: 10.1371/journal.pone.0126383. URL: https://doi.org/10.1371%5C%2Fjournal.pone.0126383.

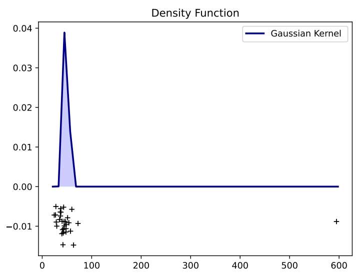
[27] user342583. *If f is differentiable in (a,b) then $\frac{1}{f}$ is differentiable at (a,b), provided $f(a,b) \neq 0$*. Mathematics Stack Exchange. URL:https://math.stackexchange.com/q/1801177 (version: 2016-05-26). eprint: https://math.stackexchange.com/q/1801177. URL: https://math.stackexchange.com/q/1801177.

[28] Yuan Yao. “Mathematics of Data III: An Introduction to Topological Data Analysis”. 2011.

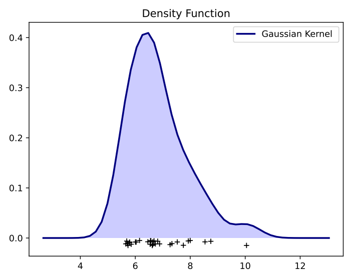
Result Figures



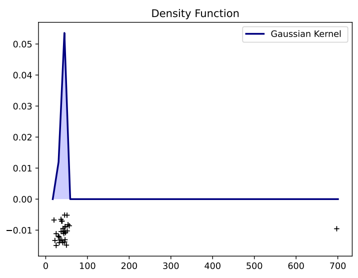
(a) Epoch 1 in two hidden layers



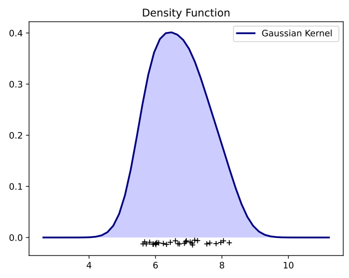
(b) Epoch 30 in two hidden layers



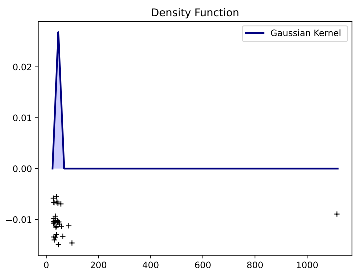
(c) Epoch 1 in three hidden layers



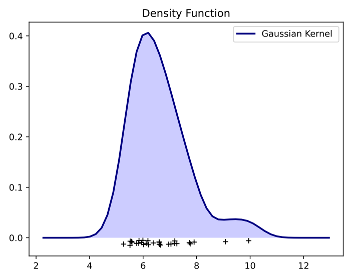
(d) Epoch 30 in three hidden layers



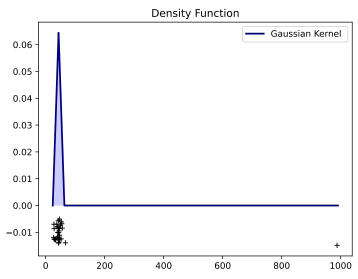
(e) Epoch 1 in five hidden layers



(f) Epoch 30 in five hidden layers

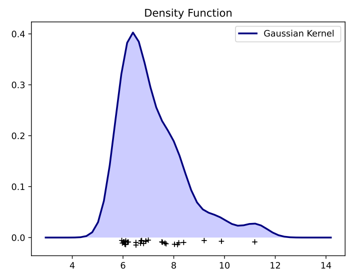


(g) Epoch 1 in ten hidden layers

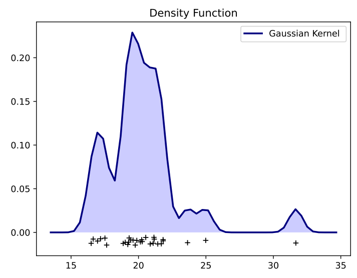


(h) Epoch 30 in ten hidden layers

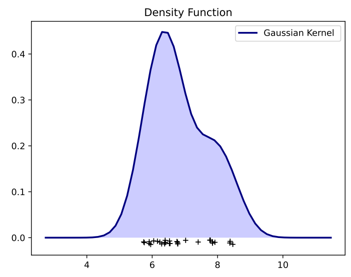
Figure 1: Density functions on persistence diagrams for CIFAR10 with total persistence over MLPs. A topological feature can be seen that is far in the distance with respect to the rest. The interesting thing is that it is visible in the four different architectures. The X-axis of the graphs are of different ranges.



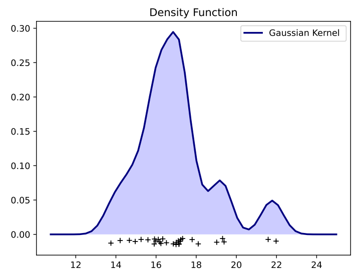
(a) Epoch 1 in two hidden layers



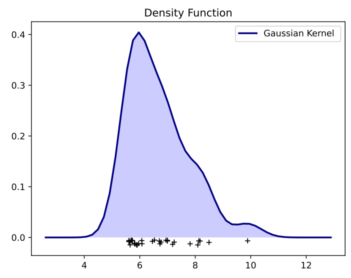
(b) Epoch 30 in two hidden layers



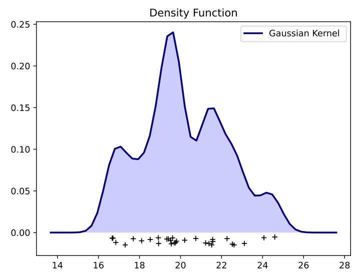
(c) Epoch 1 in three hidden layers



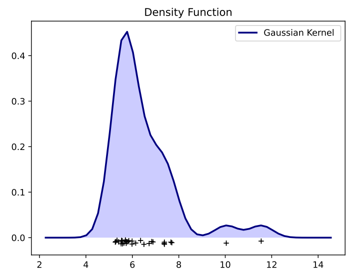
(d) Epoch 30 in three hidden layers



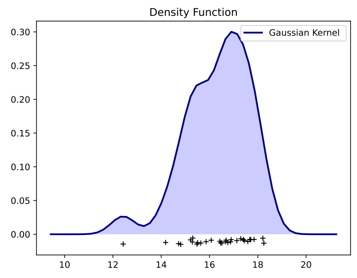
(e) Epoch 1 in five hidden layers



(f) Epoch 30 in five hidden layers

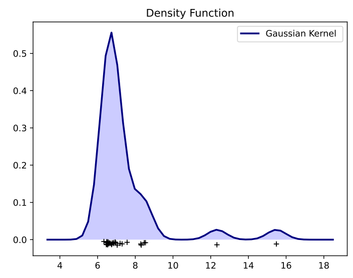


(g) Epoch 1 in ten hidden layers

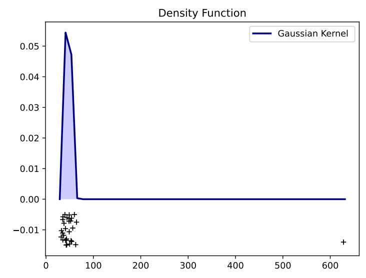


(h) Epoch 30 in ten hidden layers

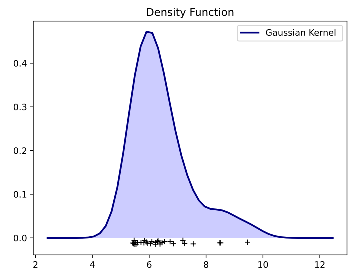
Figure 2: Density functions on persistence diagrams for CIFAR10 with group persistence over MLPs. Note that the more neurons the network has, the better it clusters the topological features of the network. It is interesting to note that in architecture 10 there is a unique clustering between the points. This is discussed in more detail in the section on cluster persistence results. The X-axis of the graphs are of different ranges.



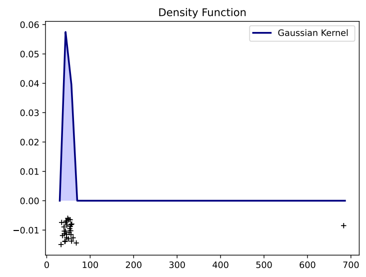
(a) Epoch 1 in two hidden layers



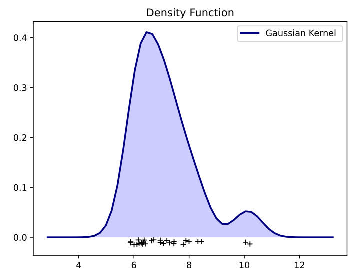
(b) Epoch 30 in two hidden layers



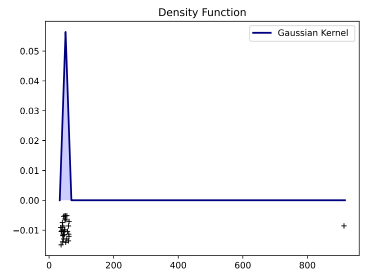
(c) Epoch 1 in three hidden layers



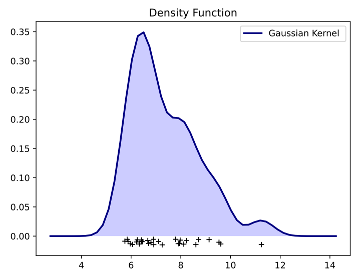
(d) Epoch 30 in three hidden layers



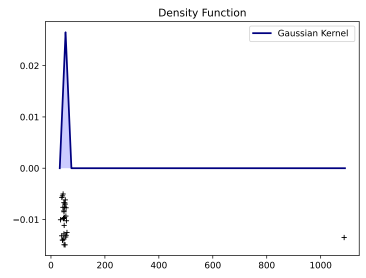
(e) Epoch 1 in five hidden layers



(f) Epoch 30 in five hidden layers

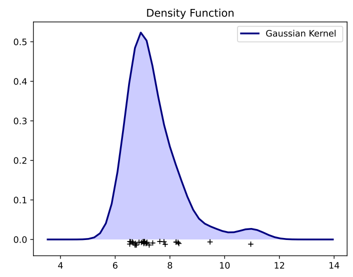


(g) Epoch 1 in ten hidden layers

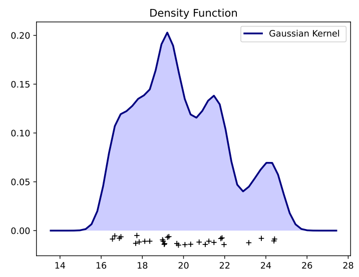


(h) Epoch 30 in ten hidden layers

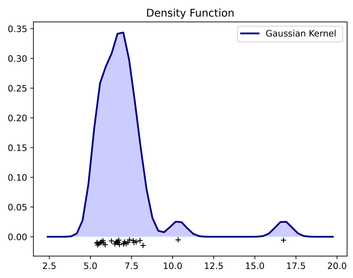
Figure 3: Density functions on persistence diagrams for CIFAR100 with total persistence over MLPs. The points of the persistence diagram have a similar behaviour as in CIFAR10. The X-axis of the graphs are of different ranges.



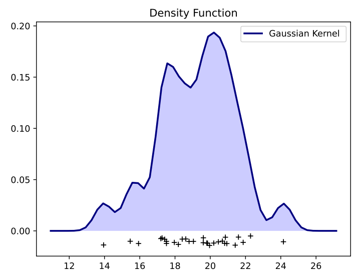
(a) Epoch 1 in two hidden layers



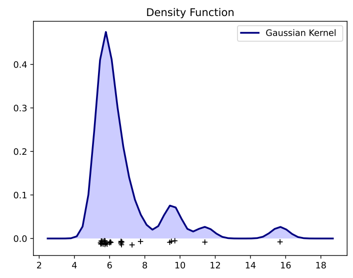
(b) Epoch 30 in two hidden layers



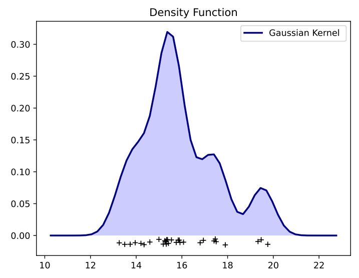
(c) Epoch 1 in three hidden layers



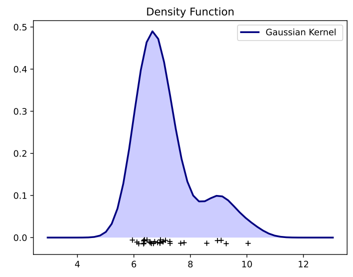
(d) Epoch 30 in three hidden layers



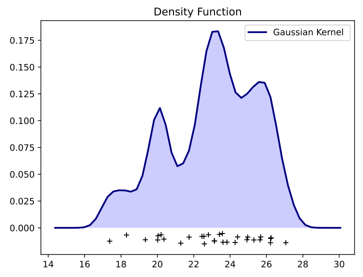
(e) Epoch 1 in five hidden layers



(f) Epoch 30 in five hidden layers

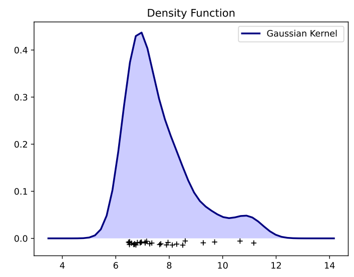


(g) Epoch 1 in ten hidden layers

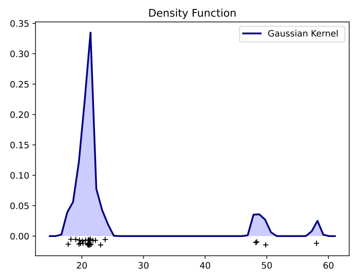


(h) Epoch 30 in ten hidden layers

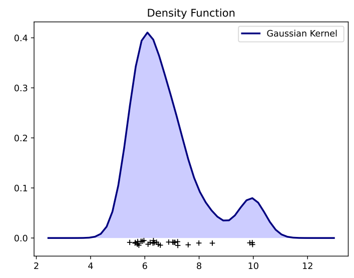
Figure 4: Density functions on persistence diagrams for CIFAR100 with group persistence over MLPs. In contrast to CIFAR10 and MNIST, a single cluster is not generated between the points of the diagram. The X-axis of the graphs are of different ranges.



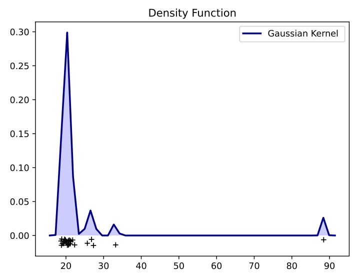
(a) Epoch 1 in two hidden layers



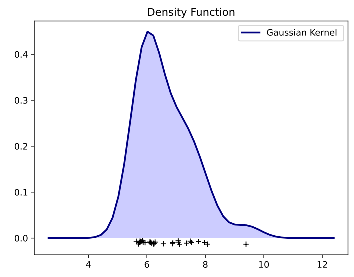
(b) Epoch 30 in two hidden layers



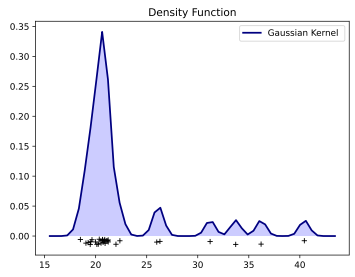
(c) Epoch 1 in three hidden layers



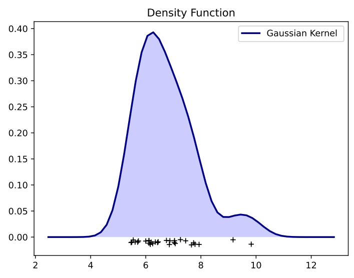
(d) Epoch 30 in three hidden layers



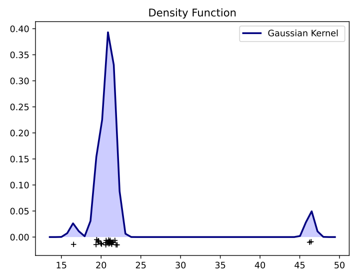
(e) Epoch 1 in five hidden layers



(f) Epoch 30 in five hidden layers

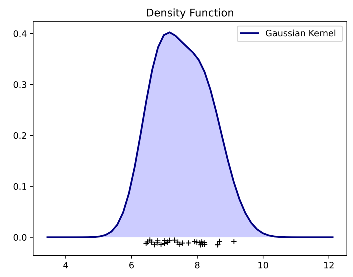


(g) Epoch 1 in ten hidden layers

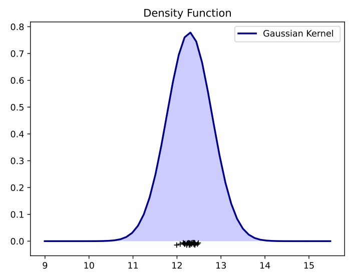


(h) Epoch 30 in ten hidden layers

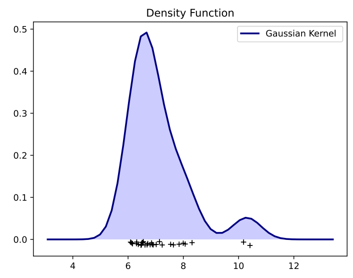
Figure 5: Density functions on persistence diagrams for MNIST with total persistence over MLPs. Although the points are farther away from the rest, it can be seen that the distance is not as long as in CIFAR10 or CIFAR100. The X-axis of the graphs are of different ranges.



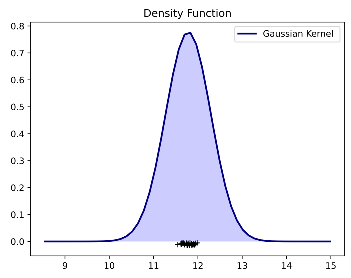
(a) Epoch 1 in two hidden layers



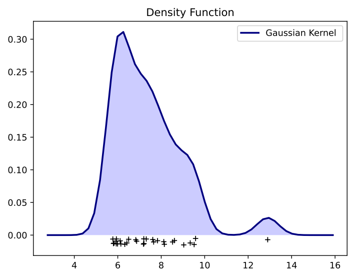
(b) Epoch 30 in two hidden layers



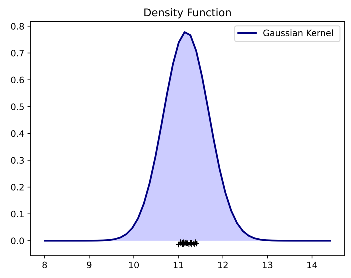
(c) Epoch 1 in three hidden layers



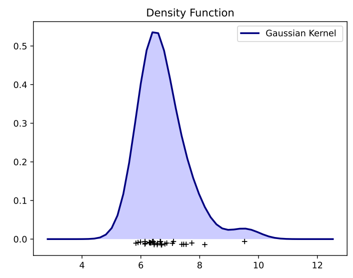
(d) Epoch 30 in three hidden layers



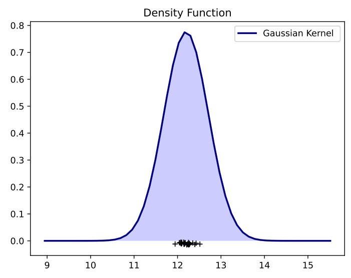
(e) Epoch 1 in five hidden layers



(f) Epoch 30 in five hidden layers



(g) Epoch 1 in ten hidden layers



(h) Epoch 30 in ten hidden layers

Figure 6: Density functions on persistence diagrams for MNIST with group persistence over MLPs. Interesting behaviour that regardless of the architecture all the points of the diagram end up in a single cluster. The X-axis of the graphs are of different ranges.