UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA "TULLIO LEVI-CIVITA"

Corso di Laurea Magistrale in Matematica

Machine Learning explainability through Persistent Homology

Relatore:
Prof. Fabio Aiolli
Correlatore:
Luca Bergamin, MSc

Laureando: Ivan Gridelli
Matricola: 2028121

Anno Accademico 2022/2023

15/12/2023

Elapsam semel occasionem
non ipse potest
Iuppiter reprehendere.

Contents

Introduction

In the last years artificial intelligence and, more specifically, machine learning algorithms have completely revolutionized the way we deal with data and are becoming indispensable tools across various sectors. Among the field of machine learning models, neural networks stand out for their ability to learn intricate patterns from data and generalizing astoundingly well as though they were mimicking the complexities of the human brain. Despite their success, understanding the internal mechanisms of neural networks still poses a considerable challenge, and the reason for their great generalizing capabilities still remains a mystery as formal measures for assessing such information have yet to be identified [34]. Previous approaches for improving theoretical and practical comprehension focus on interrogating networks with input data [33][28][29]. This thesis aims to take a different approach by leveraging topological information. In recent years, due to the growing availability of large amounts of data, Topological Data Analysis developed as a useful mathematical theory for analyzing it, benefiting from the dimensionality reduction guaranteed by topology. In particular, among Topological Data Analysis tools, that of Persistent Homology has been receiving more and more interest. In its essence, homology is concerned with the identification and characterization of holes and voids within shapes. Persistent homology takes this concept further by examining the lifespan of these topological features as they persist across different scales. As can be seen in figure 1 persistent homology is able to capture the 'overall' information in our data, in a stable way even in presence of some amounts of noise. In particu-

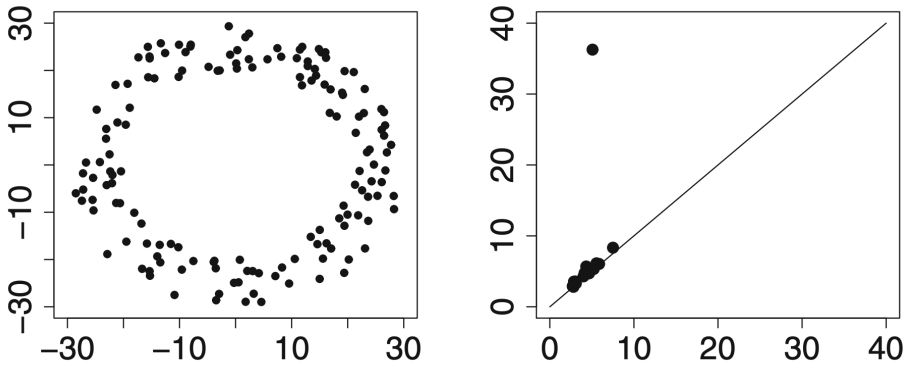


Figure 1: On the left a point cloud dataset and on the right we can see its 1-dimensional persistence diagram. Persistence diagrams are summaries of relevant information captured by the persistent homology groups. The presence of a point much far from the $y = x$ line indicates the presence of a 1 dimensional component (hole) that persists more than the other components.

lar, Persistent Homology has recently been successfully used in the analysis of graphs and network structures with promising results. [18] [1] The literature dedicated to the study of the graph structure naturally associated to a neural network is yet quite scarce and most importantly lacks a common mathematical foundation. The aim of this thesis is to provide a formal setting for analyzing the neural network structure using persistent homology whilst taking into consideration the need for efficiency that is needed for the application of this tool. In particular we are interested in understanding a network's generalizing ability without interrogating it on a portion of the dataset that was withheld for this specific purpose. Such result in a context of data scarcity could prove to be an astounding feat. The use of persistent homology has three main advantages *per se*: it has a solid theoretical foundation, it is efficiently computable and it is robust to small perturbations[23]. Since we can naturally associate a weighted graph to a neural network, and to that a weighted simplicial complex, we can develop in a natural way a filtration that allows the computation of its persistent homology. Moreover, in papers as [29] the authors were able to detect statistical interactions of any

order or form captured by a feedforward neural network by examining *only* its weight matrices, suggesting that posing an emphasis on the analysis of the evolution of the weights could be a key factor in understanding the neural network learning process. Moreover, recently, Persistent homology has been used for weighted network classification as in [27] and in [19] contributing as seminal work. As also the authors note in both articles, persistent homology could yield interesting results in the field of machine learning. Neural networks work as knowledge distilling pipelines, meaning that the degree of feature abstraction increases with the depth of neural networks layers. For example, images of cats are incrementally abstracted from pixels to diagonal lines and ear shapes. [31] Additionally, neural networks can detect cats based on feature combinations. A deep neural network effectively acts as an information distillation pipeline, with raw data going in and being repeatedly transformed so that information deemed irrelevant is filtered out, and useful information is magnified and refined. Feature relationships represent the implementation of knowledge in neural networks, which can be investigated from their structures. Persistent homology is a great tool that can stably capture redundancy and efficacy of information captured by a network, by analyzing the persistence of its topological features. The definition of a formal common framework, in this field of study, would also allow an easy comparison and easier interpretation of the obtained results.

Let us now go into more detail on the structure of this thesis. In chapter 1 we provide an overview of how topology has been used in the field of machine learning, and where in this classification falls our study. In chapter 2 we provide the mathematical framework necessary to define the concept of persistent homology and the necessary definitions related to weighted graphs and weighted complexes. In chapter 3 we give a formal mathematical definition of neural network and go into detail about the most relevant concepts that are needed for our considerations about networks. In chapter 4 we describe the state of the art in terms of analyzing the weighted graph associated to a neural network using persistent homology, what has been done and the

results that have been extrapolated. In chapter 5 we go into detail about our contributions in this topic. We explain the pipeline of processes we devised that can be used to study the topological properties of the network. Moreover we have proven some results that give us useful tools to advance the research on this topic. In chapter 6 we show the results of our experiments and compare them to the existing results, that have been translated into the methodology that we devised. Finally, we give an overview of our contributions and highlight what could be interesting avenues of further future research.

Chapter 1

Topological Machine Learning

Before we delve in the formal mathematical framework, let us give an overview of the fields of study that merge Topological Data Analysis and Machine Learning. In order to be a bit more concise, from now on we will often refer to Topological Data Analysis as TDA and Machine Learning as ML. In the last thirty years, Topological Data Analysis developed as a useful mathematical theory for analyzing data[11], benefiting from the dimensionality reduction guaranteed by topology. Topology can construct low-dimensional representations of high-dimensional manifolds of data, thereby reducing the dimensionality of the parameter space. On the other hand, Machine Learning techniques have shown remarkable success in tasks like classification, regression, clustering, and pattern recognition. This chapter aims to delve into the convergence of TDA and ML, a field of study that has been rapidly growing in the last years, exploring how these two fields can complement each other to enhance our ability to extract knowledge about a Neural Network's inner workings. By leveraging the topological concepts of persistence diagrams, simplicial complexes, and homology, TDA provides a means to capture the essential topological features of the Network's graph structure. As we will see, the intersection of TDA and ML has recently produced promising results in various domains, including biology, neuroscience, social sciences, and engineering. Topological ML techniques can be categorized using the following

criteria:

1. By extrinsic we mean that no analysis of the topology of the machine learning model or the neural network itself is incorporated. The topological analysis only relates to the data.

2. Intrinsic methods are those which incorporate the topological analysis of aspects of the machine learning model itself.

Moreover they can be further classified as:

1. Observational methods "observe" the topology of the data or model but they do not directly influence the model training or architecture.

2. Interventional methods apply topological properties of the data, as well as analysis of topological features of machine learning models, in order to inform the architectural design and/or model training.

We will see that most extrinsic approaches are observational, i.e., they do not inform the choice of model afterwards, while most intrinsic approaches are interventional, i.e., they result in changes to the choice of model or its architecture.

1.1 Extrinsic methods

These methods aim at suitably representing topological features in order to use them as input features for machine learning models. A large class of such methods consists of vectorisation methods that aim to transform persistent homology information into a feature vector form which is then fed to machine learning models. There are two predominant strategies for facilitating the integration of topological features into machine learning algorithms, namely

1. different representations that give rise to feature vectors,

2. kernel-based methods that allow the integration into classifiers.

The stability of such representations is based on the fundamental stability theorem proved in Stability of Persistence Diagrams by H. Edelsbrunner[8]. Let us see some examples of both.

1.1.1 Feature Vectors

Arguably the most simple form of employing topological descriptors in machine learning tasks uses summary statistics, such as the total persistence of a persistence diagram, its p-norm, or other scalar parameters. While all of these approaches result in scalar-valued summary statistics, they are often not directly applicable to complex machine learning tasks, which require more expressive representations. There are interventional methods that use Betti Functions like PLLay[20] where the authors introduce a novel topological layer for general deep learning models based on persistence landscapes, or PersLay[5] where the authors focus on persistence diagrams built on top of graphs. In figure 1.1 we can see a summary of how the PersLay model works. On the other hand there are vectorialization observational methods that introduce the concepts of Persistence Images or Persistence Landscapes as also introduced in the PLLay article by, for instance, using a differentiable projection function for persistence diagrams with learnable parameters, which demonstrates that persistence diagrams of a data set can be easily integrated into any deep learning architecture. Another example is given in [17] where the authors propose a technique that allows to input topological information into deep neural networks and learn a task-dependent optimal representation during training.

1.1.2 Kernel Methods

As an alternative to the previously discussed representations, now we want to briefly focus on persistence diagrams. The space of persistence diagrams can be endowed with metrics, such as the bottleneck distance. However, there is no natural Hilbert space structure on it, and such metrics tend

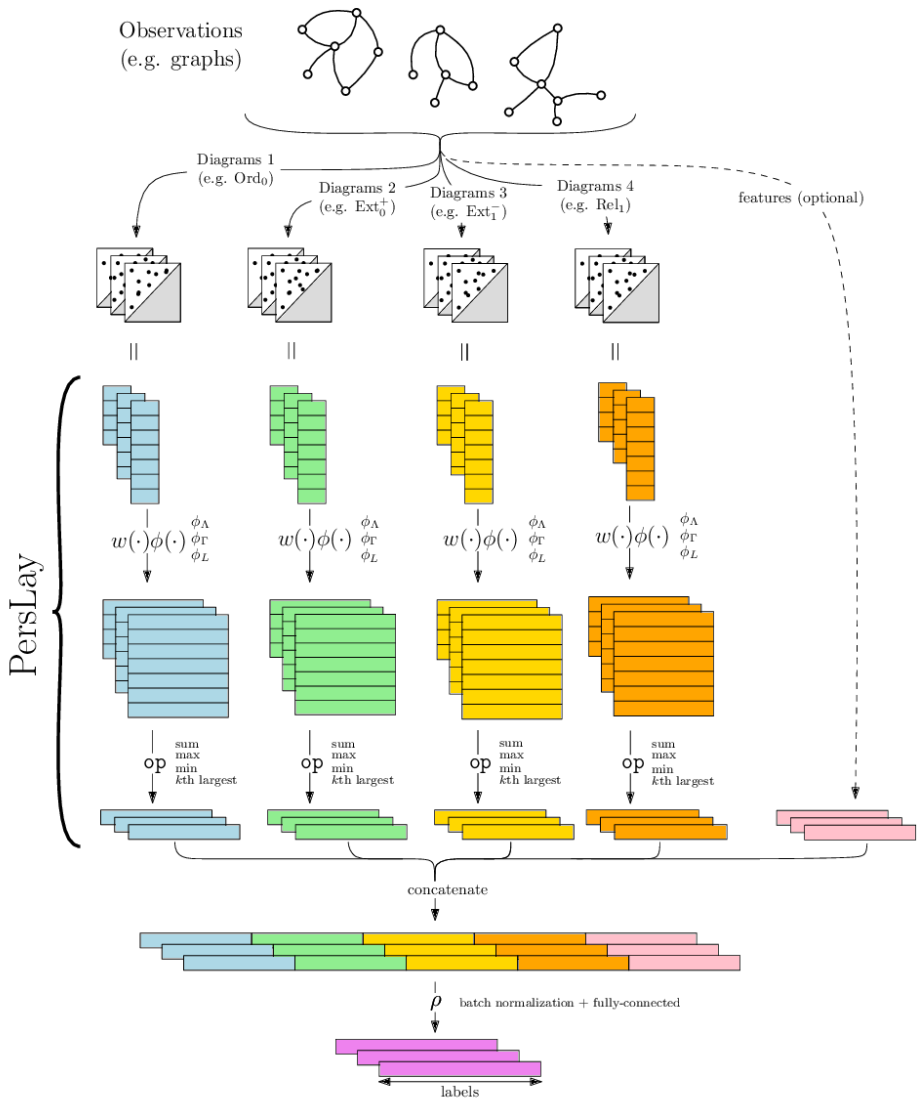


Figure 1.1: How the exstrinsic model, that uses feature vectors, called PersLay is organized.

to be computationally prohibitive or require the use of complex approximation algorithms. Kernel methods provide a way of implicitly introducing such a Hilbert space structure to which persistence diagrams can be mapped via the feature map of the kernel. By defining a kernel on the set of persistence diagrams, one obtains a vector representation via the feature map. This then allows for downstream use in machine learning models. One example can be seen in [25] where they ground their approach in a heat map applied to persistence diagrams.

1.2 Intrinsic methods

This section focuses on methods that either incorporate topological information directly into the design of a machine learning model itself, or leverage topology to study aspects of such a model. The latter will be the approach we have taken in consideration.

1.2.1 Regularization techniques

Topological information is often used for regularization: As a recent example, Moor et al. propose a topological autoencoder[22], which aims to preserve topological features of the input data in low-dimensional representations. This is achieved via a regularization term that rewards when the persistence diagrams of both the latent and input space are topologically similar. Conversely, in A Topological Regularizer for Classifiers via Persistent Homology[6] the authors develop a measure of the topological complexity (in terms of connected components) of the classification boundary of a given classifier. Said topological information is then used for regularization in order to force the topological complexity of the decision boundary to be simpler, containing fewer features of low persistence. Thus, topological information serves as a penalty during classification so that training the classifier itself can be improved.

1.2.2 Model Analysis

In a different direction, the topological analysis of the intrinsic structure of a classifier, such as a neural network, makes it possible to improve a variety of tasks and, as we said previously, this is the direction we will be focusing on. This includes the analysis of training behavior as well as model selection or architecture selection in the case of neural networks. While the literature dedicated to the better understanding of deep neural networks has typically focused on its functional properties, in [26] the authors took a different perspective to focus on the graph structure of a neural network. Specifically, the authors treat a (feedforward) neural network as a stack of bipartite graphs. From this view, the authors propose "neural persistence", a complexity measure which summarizes topological features that emerge when calculating a filtration of the neural network graph, where the filtration weights are given by the network parameters. The authors showed that neural persistence can distinguish between well-trained and badly-trained networks. This measure is oblivious to the functional behavior of the underlying network, but only focuses on its weighted structure. Nevertheless, the authors showed that it can be used for guiding early stopping solely based on topological properties of the neural network, potentially saving validation data used for the early stopping decision. A similar research has been carried out in [15]. In their study, the authors propose a novel approach to monitor the training of neural networks: instead of relying on a validation (holdout) set to estimate the model's generalization, the authors advocate for the use of Persistent Homology by examining the evolution of PH diagram distances during the neural network training process, utilizing simplicial complex representations. Multiple network architectures and datasets are considered. The findings reveal a significant correlation between the PH diagram distance measured between consecutive states of the neural network and the corresponding validation accuracy, therefore suggesting that it may be possible to intrinsically estimate a neural network's generalization error without the need for a separate validation set. Another very interesting approach is given by Carrlson et

Al. in Exposition and Interpretation of the Topology of neural networks[12], where the authors show that the weights of convolutional layers encode simple global structures which dynamically change during training of the network and correlate with the network's ability to generalize to unseen data. Moreover, the authors find that topological information on the trained weights of a network can lead to improvements in training efficiency and reflect the generality of the data set on which the training was performed. We will go more in depth regarding this topic in chapter 5.

Chapter 2

Persistent Homology

Firstly, let us define the building blocks required to reach the definition of persistent homology.

2.1 Simplexes

Definition 1. *Let us consider d+1 affinely independent points* $x_0, \ldots, x_d \in \mathbb{R}^n$. *The convex hull of those points* $s_d := conv(x_0, ..., x_d)$ *is called d-simplex.*

For instance, as can be seen in figure 2.1 a 0-simplex in $\mathbb{R}^3$ is a point, a 1-simplex is a line, a 2-simplex is necessarily a triangle, a 3-simplex is a tetrahedron and so on...

Definition 2. *If we consider a d-simplex* s_d, *given by the convex closure*

Figure 2.1: Example of simplices.

$conv(x_0, ..., x_d)$, a face of s_d is the convex closure of some of the generating points of s_d.

For example: Let us consider a 3-simplex in $\mathbb{R}^3$: $s_3 = conv\big(\big(\begin{smallmatrix}1\\0\\0\end{smallmatrix}\big), \big(\begin{smallmatrix}0\\1\\0\end{smallmatrix}\big), \big(\begin{smallmatrix}0\\0\\1\end{smallmatrix}\big), \big(\begin{smallmatrix}0\\0\\0\end{smallmatrix}\big)\big)$ This full tetrahedron has 2^4 faces, for instance a couple of them could be $conv\big(\big(\begin{smallmatrix}1\\0\\0\end{smallmatrix}\big), \big(\begin{smallmatrix}0\\0\\1\end{smallmatrix}\big), \big(\begin{smallmatrix}0\\0\\0\end{smallmatrix}\big)\big)$ or $conv\big(\begin{smallmatrix}0\\0\\0\end{smallmatrix}\big)$.

Definition 3. *A geometrical simplicial complex K is a set of simplexes of any dimension such that*

1. *if $s_n \in K$ and s_d is a face of s_n then also $s_n \in K$.*
2. *if $s_n, s_m \in K$ then their intersection is either empty or a common face of s_n and s_m.*
3. *any simplex of K cannot be face of infinitely many simplexes of K.*

The dimension of a complex K is defined as the largest dimension of its simplexes.

It is possible to give a more abstract definition of simplicial complex, which follows.

Definition 4. *An abstract simplicial complex is a set of finite non-empty sets, closed under inclusion. Formally, let X be a set, and let $\mathcal{K}$ be a collection of subsets of X. $\mathcal{K}$ is an abstract simplicial complex if it satisfies the following conditions:*

1. *If $\sigma \in \mathcal{K}$ and $\tau \subset \sigma$, then $\tau \in \mathcal{K}$.*
2. *If $\sigma \in \mathcal{K}$ and $\tau \in \mathcal{K}$, then their intersection $\sigma \cap \tau$ is also in $\mathcal{K}$.*

The elements of $\mathcal{K}$ are called *simplices*, and the dimension of a simplex σ is defined as $|\sigma| - 1$, where $|\sigma|$ is the number of elements in σ. The *dimension* of the abstract simplicial complex $\mathcal{K}$ is the maximum dimension of its simplices. Sometimes a simplicial complex of dimension d will be called simplicial $d-$complex. An example of what is and what is not a simplicial complex can be seen in figure 2.2. We will only be considering finite simplicial complexes.

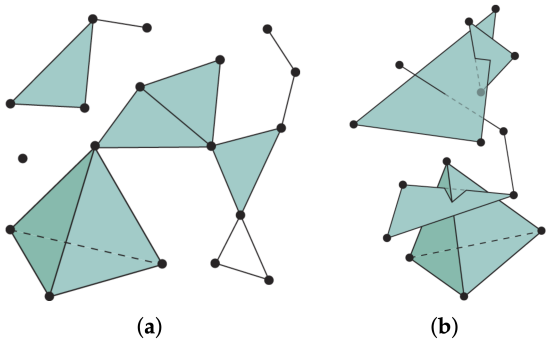


Figure 2.2: On the left (a) is a simplicial complex whereas on the right (b) is not a simplicial complex as it doesn't respect the closedness under intersection.

2.1.1 Weighted graphs

We are interested in feedforward neural networks, to which we can naturally associate a structure as weighted graphs, hence why we need the following definitions. Firstly, let us define what a weighted simplicial complex is:

Definition 5. *A weighted simplicial complex is a pair (K, w) where K is a simplicial complex and w is a function $w : K \to \mathbb{R}$.*

Definition 6. *We will say that w is monotone if $w(\sigma) \leq w(\tau)$ for all $\sigma \subseteq \tau \in K$.*

Since we want to consider the simplicial 1-complex associated to a weighted graph we need the following definitions:

Definition 7. *A weighted graph is a tern $G_w = (V, E, w)$ where (V, E) is a graph and w is a function $w : E \to \mathbb{R}$.*

We will only be considering undirected graphs meaning graphs such that $(u, v) \in E \implies (v, u) \notin E$. Moreover we will always assume that our graphs do not present self-loops. To such graphs we can naturally associate a simplicial complex in the following way

Definition 8. *The simplicial complex $K(G)$ associated to the graph G is formed by taking all non-empty subsets of V and all non-empty subsets of E and considering them as simplices. Specifically, any $v \in V$ is a 0-simplex in $K(G)$, and any $e \in E$ is a 1-simplex in $K(G)$. Formally $K(G) = \{\{u, v\}$ for $(u, v) \in E\} \cup \{v$ for $v \in V\}$.*

This definition clearly respects the inclusion requirements to be a simplicial complex. Moreover, for a weighted graph, we can define an induced weight function on $K(G_w)$, which is defined as follows.

Definition 9. *Given a (undirected) weighted graph G_w the induced weighted complex is $(K(G), \bar{w})$ where $\bar{w} : K(G) \to \mathbb{R}$, defined as*

$$\begin{cases} \bar{w}(v) = 0 \text{ for every 0-simplex of } K. \\ \bar{w}(u, v) = w(u, v) \text{ for every 1-simplex of } K. \end{cases}$$

Please note that there are multiple possible definitions for the induced weight function, and the chosen one is based on our objective which is considering the weighted simplicial complex associated to the weighted graph that represents a neural network. We will now give the abstract definition of homology, determine the simplicial homology groups associated to a complex K and then we will be ready to introduce the definition of persistent homology, the main tool we will be using in our study.

2.2 Homology

Definition 10. *We will call chain the following pair $G = \left(\{G_j\}_{j\in\mathbb{Z}}, \{\partial_j\}_{j\in\mathbb{Z}}\right)$ where each G_k is a commutative group and each $\partial_j : G_j \to G_{j-i}$ is a group homomorphism called $j^{th}-$border operator. The border operators must satisfy $\partial_{j-1} \circ \partial_j = 0$.*

One could visualize a chain as the following sequence:

$$\ldots \xleftarrow{\partial_{j-1}} G_{j-1} \xleftarrow{\partial_j} G_j \xleftarrow{\partial_{j+1}} G_{j+1} \xleftarrow{\partial_{j+2}} \ldots$$

Definition 11. *Given a chain G we will call*

- *k-cycles elements of the group* $Z_k(G) := \ker(\partial_k) \subseteq G_k$.
- *k-borders elements of the group* $B_k(G) := Im(\partial_{k+1}) \subseteq G_k$.

Notice how the property $\partial_{j-1} \circ \partial_j = 0$ implies $B_k(G) \subseteq Z_k(G)$.

Definition 12. *Given a chain G we will call* $k^{th}-$*homology group of the chain G the quotient*

$$H_k(G) := Z_k(G)/B_k(G)$$

. Elements of $H_k(G)$ *are called homology classes.*

Intuitively one could think that homology classes are 'surviving' cycles, i.e. cycles that are not borders of anything.

Let us recall that any finitely generated commutative group G has a decomposition as $G = \overset{r}{\oplus}\mathbb{Z} \oplus \mathbb{Z}_{i_1} \oplus \cdots \oplus \mathbb{Z}_{i_s}$. The number r is called rank of the group G.

Definition 13. *Given a chain G we will call* $k^{th}-$*Betti number* β^k *the rank of the group* $H_k(G)$.

2.3 Simplicial Homology

Given any topological space X, we could define its associated chain $G(X)$. Then, the associated homology groups are the tool used to formalize the notion of k-dimensional holes of X. For the purpose of introducing persistent homology, and for the scope of this thesis, we will just define the homology groups associated to a complex K. Thus our first objective is to define a chain associated to K, for which we need to define the commutative groups and the boundary operators.

Definition 14. *Let K be a simplicial complex. We represent with* $S_i(K)$ *the free commutative group generated by the i-dimensional simplexes. Hence elements of* $S_i(K)$ *are formal linear combinations, with coefficients in* $\mathbb{Z}$*, of all simplexes of K of dimension i.*

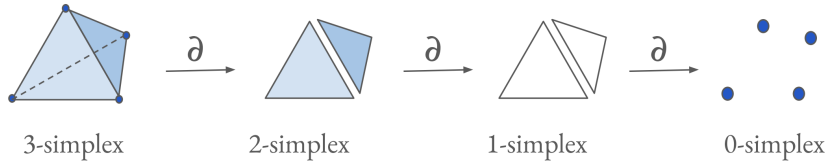


Figure 2.3: An example of how the boundary operator works, it associates to a $d-$simplex the formal sum of its $(d-1)-$simplices.

Notice how the only possibly non-empty groups are the ones indexed from 0 to the dimension of the complex.
Let us now define the boundary operator $\partial_i : S_i(K) \to S_{i-1}(K)$:

Definition 15. *The border of an i-dimensional simplex* $s = conv(x_0 \dots x_i)$ *is defined as* $\partial_i(s) := \sum_{k=0}^{i} (-1)^k conv(x_0 \dots \hat{x}_k \dots x_i)$ *where the hat denotes that the point has been removed. The border is then linearly extended to all elements of* $S_i(K)$.

Notice how the composition of two borders has to be zero. In the context of abstract simplicial complexes, the boundary operator, denoted as ∂, is used to define the boundary of simplices. For a simplex σ in the complex, $\partial(\sigma)$ is a formal sum of its $(k-1)$-dimensional faces, where k is the dimension of σ.

The orientation of an abstract simplicial complex is given as follows:

Definition 16. *An orientation of a simplex* σ *in an abstract simplicial complex is a choice of ordering for its vertices. More formally, for a simplex* σ *with vertices* $\{v_0, v_1, \dots, v_k\}$, *an orientation specifies a permutation of these vertices, which determines a consistent ordering.*

The choice of orientation is typically represented as a signed permutation, and the sign indicates whether the orientation is "positive" or "negative." The sign depends on the parity (even or odd) of the permutation. An orientation of a simplex induces orientations of its faces. If σ is a k-dimensional simplex,

and τ is one of its k-dimensional faces, then the orientation of σ induces an orientation on τ by considering the ordering of vertices in τ based on the ordering in σ.
We can now define the boundary operator in the case of abstract simplicial complexes:

Definition 17.
- *For a 0-dimensional simplex (vertex), $\partial(\sigma)$ is the zero chain, representing an empty boundary.*
- *For higher-dimensional simplices, $\partial(\sigma)$ is defined as the formal sum of its $(k-1)$-dimensional faces with appropriate orientations. This ensures that each face is counted with the correct sign. Mathematically, if σ is a k-dimensional simplex with vertices $\{v_0, v_1, \ldots, v_k\}$, then:*

$$\partial(\sigma) = \sum_{i=0}^{k} (-1)^i [v_0, v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_k]$$

Where i ranges from 0 to k, representing the k different $(k-1)$-dimensional faces of σ, and $[v_0, v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_k]$ represents the $(k-1)$-dimensional face obtained by removing the i-th vertex from σ.

Hence we have successfully created the chain

$$\ldots \xleftarrow{\partial_{i-1}} S_{i-1}(K) \xleftarrow{\partial_i} S_i(K) \xleftarrow{\partial_{i+1}} S_{i+1}(K) \xleftarrow{\partial_{i+2}} \ldots$$

Definition 18. *If $S(K)$ is the chain associated to K, we call $i^{th}-$ simplicial homology group the group $H_i(K) := H_i(S(K))$.*

Now, β^0 coincides with the number of connected components of the complex, β^1, its tunnels and its holes, β^2, the shells surrounding voids or cavities, and so on... For the analysis of real-world data sets, however, Betti numbers turn out to be of limited use because their representation is too unstable, hence welcoming the definition of persistent homology. Given a weighted simplicial complex K with weights $a_0 \leq a_1 \leq \ldots \leq a_{m-1} \leq a_m$, which are commonly thought to represent the idea of a scale, the filtration can be seen as the 'growth' of K as the scale is being changed. During this growth

process, topological features can be created (new vertices may be added, for example, which creates a new connected component) or destroyed (two connected components may merge into one). Persistent homology tracks these changes and represents the creation and destruction of a feature as a point $(a_i, a_j) \in \mathbb{R}^2$ for indices $i \leq j$ with respect to the filtration. The collection of all points corresponding to d-dimensional topological features is called the d-th persistence diagram D_d which can be seen as a collection of Betti numbers at multiple scales. Given a point $(x, y) \in D_d$, the quantity $\text{pers}(x, y) := |y-x|$ is referred to as its persistence. Typically, high persistence is considered to correspond to features, while low persistence is considered to indicate noise [10].

2.3.1 Persistent Homology

We are now ready to define persistent homology.

Definition 19. *A filtration of a simplicial complex K is a finite sequence of (sub)complexes of K in ascending order $\emptyset = K^0 \subseteq K^1 \subseteq \dots K^{m-1} \subseteq K^m = K$.*

In the special case of weighted simplicial complexes we will consider the 'sublevel set filtration' defined as:

Definition 20. *Let (K, w) be a weighted simplicial complex with $w \in [0, 1]$ and $0 = w_1 \leq w_2 \leq ... \leq w_n = 1$ be a sequence of numbers. Then the associated sublevel set filtration is $\emptyset = K^0 \subseteq K^1 \subseteq \dots K^{n-1} \subseteq K^n = K$ where $K^0 := \emptyset$ and $K^i := \{\sigma \in K | w(\sigma) \leq w_i\}$.*

Definition 21. *For $b, d \in \{0 \dots m\}$ we call the $(b, d)-$persistent $i^{th}-$simplicial homology group $H_i^{(b,d)}(K) := Im(L_i^{(b,d)})$ where $L_i^{(b,d)}$ is the linear map $L_i^{(b,d)} : H_i(K^b) \to H_i(K^d)$ induced by the inclusion $K^b \hookrightarrow K^d$.*

While homology captures cycles in a shape by factoring out the cycles that are actually boundaries of something, persistent homology allows for the retrieval of cycles that are non boundary elements in a certain step of

the filtration but that will turn into boundaries in some subsequent step. The persistence of a cycle during the filtration gives quantitative information about the relevance of the cycle itself for the shape.

Definition 22. *Now, let γ be a class in $H_p(K_i) =: H_p^i$. We say that γ is born in K_i if $\gamma \notin H_p^{i-1,i}$. Let $f_{i,j}^p : H^p(K_i) \longrightarrow H^p(K_j)$ be the induced homomorphism by the inclusion $K_i \subseteq K_j$. On the other hand, if γ is born in K_i, we say that it dies entering K_j if $f_p^{i,j-1}(\gamma) \notin H_p^{i-1,j-1}$ but $f_p^{i,j}(\gamma) \in H_p^{i-1,j}$, which means that γ either merges with another "older" class or is "killed" as the boundary of a chain.*

The couple of birth and death of the $p-$dimensional class γ (i, j) is called persistence pair. In these terms, we observe that when two classes merge, the one that survives is always the one that was born first. Furthermore, taking into account the last definition, we have that the persistent homology group $H_p^{i,j}$ is formed by the homology classes of K_i that are still alive in K_j, and $\beta_p^{i,j}$ precisely counts these classes. Formally, we have:

$$H_p^{i,j} = \frac{Z_p(K_i)}{B_p(K_j) \cap Z_p(K_i)}$$

Example

Let us see a simple example to better illustrate how persistence pairs are evaluated. Let us consider the simplicial 2-complex K of figure 2.4, and the filtration of figure 2.5. Since this is a 2-complex we will have only three possibly non empty homology groups. Let us track the free commutative groups associated to each complex of the filtration in table 2.1. In table 2.2 see what are the simplicial homology groups associated to each chain

$$\emptyset \xleftarrow{\partial_0} S_0(K_i) \xleftarrow{\partial_1} S_1(K_i) \xleftarrow{\partial_2} S_2(K_i) \xleftarrow{\partial_3} \emptyset$$

For instance, what happens from $H_0(K^1)$ to $H_0(K^2)$ is the fact that since all vertices are elements of $Ker(\partial_0)$ but not boundaries of anything ($Im(\partial_1) = \emptyset$) they are all generators. However, in $H_0(K^2)$ we have that $\partial_1(AB) = A + B$

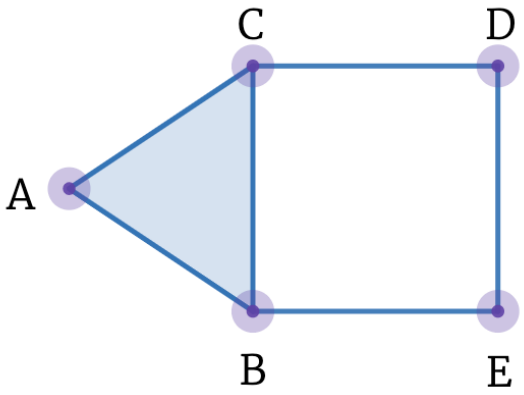


Figure 2.4: The simplicial complex K.

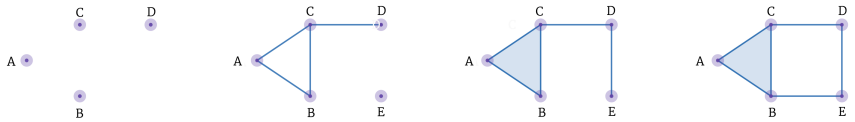


Figure 2.5: A filtration associated to our simplicial complex. $K^0 = \emptyset$ is not shown, but we have, starting from the left, K^1, K^2, K^3 and $K^4 = K$.

hence in the quotient $A = B$ so B is no longer a generator and merges with the class A. The same happens for C and D. Moreover, $H_1(K^2)$ has dimension 1 since $\partial_1(AB - BC + CA) = A - B + B - C + C - A = 0$ hence that is a cycle but $Im(\partial_2) = \emptyset$ hence it is not boundary of anything. However, this cycle dies at $H_1(K^2)$ since it is the boundary of $ABC \in S_2$. As we can see from the terms we have use it becomes natural to think about the birth and death (or merging) of the homology classes. By tracking the births and death of component we can get the three multisets of persistence pairs (which summarize the persistent homology groups) that can be seen in table 2.3.

2.3.2 Vietoris-Rips homology

In the particular case of a metric space X one of the most used tools in TDA is the Vietoris-Rips complex filtration. Let X be a metric space with

Simplices	S_0	S_1	S_2
K^0	$\emptyset$	$\emptyset$	$\emptyset$
K^1	$< A, B, C, D >$	$\emptyset$	$\emptyset$
K^2	$< A, B, C, D, E >$	$< AB, BC, CA, CD >$	$\emptyset$
K^3	$< A, B, C, D, E >$	$< AB, BC, CA, CD, DE >$	$< ABC >$
K^4	$< A, B, C, D, E >$	$< AB, BC, CA, CD, DE, BE >$	$< ABC >$

Table 2.1: Table with the associated groups to each simplicial complex.

Generators	H_0	H_1	H_2
K^0	$\emptyset$	$\emptyset$	$\emptyset$
K^1	$\{[A], [B], [C], [D]\}$	$\emptyset$	$\emptyset$
K^2	$\{[A], [E]\}$	$\{[AB - BC + CA]\}$	$\emptyset$
K^3	$\{[A]\}$	$\emptyset$	$\emptyset$
K^4	$\{[A]\}$	$\{[CD - DE + EB - BC]\}$	$\emptyset$

Table 2.2: Table with the generators of the homology groups associated to each simplicial complex of the filtration.

metric $d : X \times X \to \mathbb{R}$, and let $r \geq 0$. The Vietoris-Rips complex $\mathrm{VR}_r(X)$ is defined as follows:

1. **Vertex Set:** The vertex set of $\mathrm{VR}_r(X)$ is the set of points in X.

2. **Simplex Inclusion:** For each subset $S \subseteq X$ such that the pairwise distances between points in S are all less than or equal to r, include the simplex spanned by the points in S in $\mathrm{VR}_r(X)$. The simplex can be of any dimension, including 0 (vertices), 1 (edges), 2 (triangles), and so on.

Hence, formally, the Vietoris-Rips complex $\mathrm{VR}_r(X)$ is then given by:

$$\mathrm{VR}_r(X) = \{\sigma \subseteq X \mid \forall p, q \in \sigma : d(p, q) \leq r\}$$

Here, σ represents a simplex in the complex, and the condition $d(p, q) \leq r$ ensures that the simplex is included if and only if the pairwise distances

Persistence Pairs	D_0	D_1	D_2
Filtration of K	$\{(1,2),(1,2),(1,2),(2,3),(1,+\infty)\}$	$\{(2,3),(4,+\infty)\}$	$\emptyset$

Table 2.3: Table with the persistence pairs for dimensions 0, 1 and 2.

between its vertices are less than or equal to r. Now since if $r \leq s \in [0,\infty)$, there is a simplicial inclusion map

$$\iota_{r,s} : \mathrm{VR}_r(X) \hookrightarrow \mathrm{VR}_s(X).$$

Then we can define the Vietoris-Rips filtration as the nested collection of the following complexes:

$$\mathrm{VR}(X) = \{\mathrm{VR}_r(X)\}_{r\in[0,diam(X))}.$$

2.4 Representations of persistence

Let us reiterate the meaning of the terms birth and death. An i-dimensional homology class σ is born at K^b if it is not in the image of the map induced by the inclusion $K^{b-1} \subseteq K^b$ i.e. in $H_i^{(b-1,b)}(K)$. Furthermore, if σ is born at K^b it dies entering K^d if the image of the map induced by $K^{b-1} \subseteq K^{d-1}$ does not contain the image of σ but the image of the map induced by $K^{b-1} \subseteq K^d$ does. Formally, σ is born at K^b and dies in K^d if σ belongs to $H_i^{(b-1,d-1)}(K)$ but not to $H_i^{(b-1,d)}(K)$. We say that σ is born at time b, died at time d and persisted for $d-b$. We can represent these information using persistence diagrams, one for each dimension.

Definition 23. *The $i^{th}-$persistence diagram is the collection of couples (b,d) tracking the times of birth and death of every i-dimensional homology classes. Each diagram is now a multiset since classes can be born simultaneously and they can die simultaneously. The rank of the image of a map $L_i^{(b,d)} : H_i(K^b) \to H_i(K^d)$ i.e. the Betti number of $H_i^{(b,d)}(K)$ is the number of i-dimensional homology classes that are born at or before K^b and are still alive at K^d . This includes the essential classes of K, meaning the ones that do*

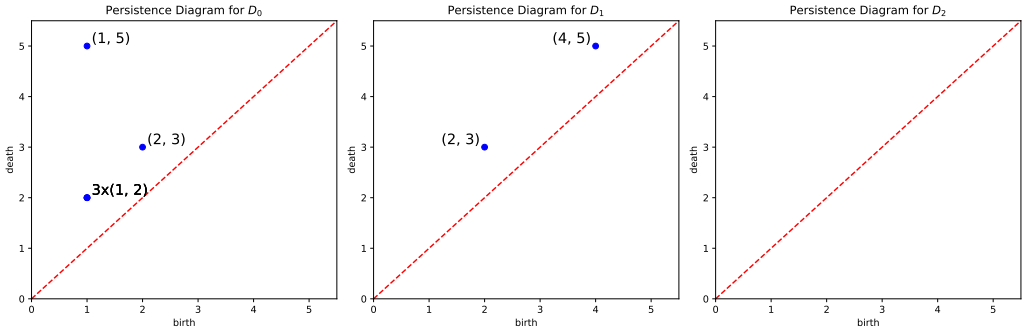


Figure 2.6: Persistence diagrams of dimensions 0,1 and 2 associated to the filtration of the complex K.

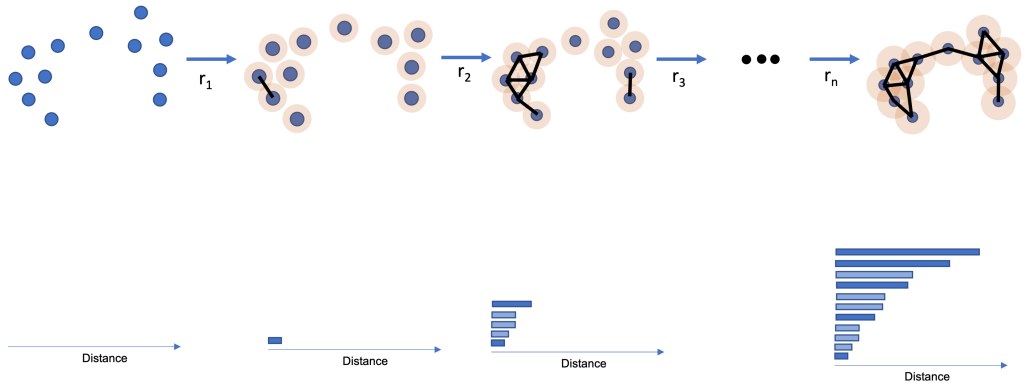


Figure 2.7: An example of how persistence barcodes are evaluated for a Vietoris-Rips complex filtration.

not die within the filtration. It is convenient to represent an essential class born at K^b by the point (b, ∞) in the diagram.

For instance, in figure 2.6 are the persistence diagrams associated to the example of the previous section.

Other representations for persistent homology features are persistence barcodes, which can be seen in figure 2.7, whose main goal is the one of highlighting couples with the longest persistence. Finally, another equivalent representation for persistence diagrams is the one of persistent landscapes, so called for their similarity to a mountainview. They are equivalent to

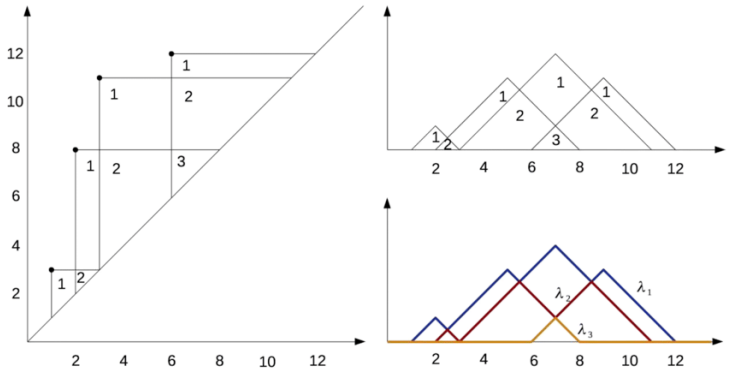


Figure 2.8: An example of persistence diagram on the left, and the corresponding persistent landscape on the right.

persistence diagrams, but rotated at a 45 degree angle as can be seen in figure 2.8.

Chapter 3

Neural networks

A neural network is a computational model inspired by the behavior of biological neurons, the basic building blocks of the brain. It consists of interconnected artificial neurons or nodes organized into layers. Each neuron processes information, performs mathematical operations on input data, and communicates the results to other neurons. This communication occurs through weighted connections between neurons. A node (or neuron) in a neural network typically has the following characteristics:

- **Input**: Nodes receive input from other nodes or external sources, and they process this input as part of the network's computation.
- **Weights and Biases**: Nodes are associated with weights and biases, which determine the strength of connections with other nodes and introduce flexibility into the node's behavior. Weights and biases respectfully are the parameters that allow an affine transformation of the input.
- **Activation Function**: Nodes often apply an activation function to the weighted sum of their inputs, introducing non-linearity and enabling the node to model complex relationships in the data.
- **Output**: The result of a node's computation, after applying the activation function, is transmitted as output to other nodes in subsequent

layers of the neural network.

An activation function, denoted as f, is a mathematical function that operates on the weighted sum of inputs and introduces non-linearity into the neural network's computations. It is applied to the output of each neuron (or node) in the network, shaping the neuron's output based on the weighted input. Activation functions are essential because they determine whether a neuron should be activated (fired) or not, and to what degree. Mathematically, the activation function takes the weighted sum z as input and produces the neuron's output a:

$$a = f(z)$$

Activation functions serve multiple purposes in neural networks, including adding non-linearity, enabling the network to learn complex functions, and introducing sparsity[16]. Each activation function has its unique characteristics and use cases. Let's introduce some of the most commonly used activation functions:

- Sigmoid Activation (Logistic): The sigmoid function maps the input to a range between 0 and 1.

$$f(z) = \frac{1}{1 + e^{-z}}$$

- Hyperbolic Tangent (Tanh): Tanh is similar to the sigmoid but maps the input to a range between -1 and 1. It has often been used in hidden layers.

$$f(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

- Rectified Linear Unit (ReLU): ReLU is a popular choice for hidden layers and has the advantage of not suffering the Vanishing Gradient Problem[13]. It sets all negative inputs to zero and leaves positive inputs unchanged.

$$f(z) = \max(0, z)$$

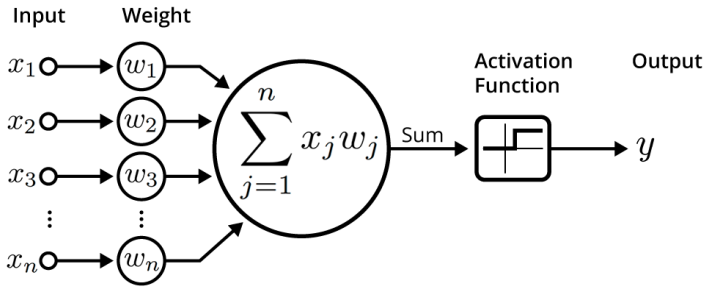


Figure 3.1: An illustration of the construction of a single neuron, without bias.

The choice of an activation function often depends on the specific task and the architecture of the neural network. In figure3.1 we can see how all these notions come together in building a single neuron.

3.1 Multi layer network with bias

Let us mathematically define a feedforward q-layer network with bias terms.[21] We talk about feedforward neural networks in cases where there are no loops, and recurrent neural networks in cases where at least one loop is present.

- Inputs: $x = [x_1, x_2, \ldots, x_n]^T$;
- Outputs: $y = [y_1, y_2, \ldots, y_m]^T$;
- Input Layer, $j = 1$: $neuron(1, i)$, where $i = 1, 2, \ldots, p_1$, with generic weight $\gamma_{ik}^{(1)}$ and bias $b_i^{(1)}$, where $k = 0, 1, 2, \ldots, n$. In matrix form, denoting $f(\cdot)$ as the activation function, $w_i^{(1)} = [\gamma_{ik}^{(1)}]^T$, and $W^{(1)} = [w_i^{(1)}]$:

$$\begin{cases} s^{(1)} = W^{(1)}u + b^{(1)} \\ z^{(1)} = f(s^{(1)}) \end{cases}$$

- Intermediate Layers, $1 < j < q$: $neuron(j, i)$, where $i = 1, 2, \ldots, p_j$ with generic weight $\gamma_{ik}^{(2)}$ and bias $b_i^{(j)}$, where $k = 0, 1, 2, \ldots, p_{j-1}$. In matrix form, with $f(\cdot)$ as the activation function, $w_i^{(j)} = [\gamma_{ik}^{(2)}]^T$, and $W^{(j)} = [w_i^{(j)}]$:

$$\begin{cases} s^{(j)} = W^{(j)} z^{(j-1)} + b^{(j)} \\ z^{(j)} = f(s^{(j)}) \end{cases}$$

- In the output layer, denoted by $j = q$, we have $neuron(q, i)$ for $i = 1, 2, \ldots, p_q$ with generic weight $\gamma_{ik}^{(q)}$ and bias $b_i^{(q)}$, where $k = 0, 1, 2, \ldots, p_{q-1}$. In matrix form, using the activation function $f(\cdot)$, we define $w_i^{(q)} = [\gamma_{ik}^{(q)}]^T$, and $W^{(q)} = [w_i^{(q)}]$:

$$\begin{cases} s^{(q)} = W^{(q)} z^{(q-1)} + b^{(q)} \\ y = f(s^{(q)}) \end{cases}$$

  The output layer processes the information and computes the final output y.

The overall input-output relationship in the neural network with bias can be represented as:

$$y = f\left(W^{(q)}\left(\ldots f\left(W^{(1)}x + b^{(1)}\right)\ldots\right) + b^{(q)}\right) \in \mathbb{R}^m$$

This equation describes the relationship between the network's inputs and outputs, where m represents the dimension of the output. In figure 3.2 we can see the global structure of a neural network.

3.2 Training a Neural Network

Training a neural network is the process of making it learn from data. This involves presenting the network with a labeled dataset, adjusting the weights and biases, and minimizing the difference between the predicted outputs and the ground truth labels. Training is typically performed using optimization algorithms, such as gradient descent, which update the weights to

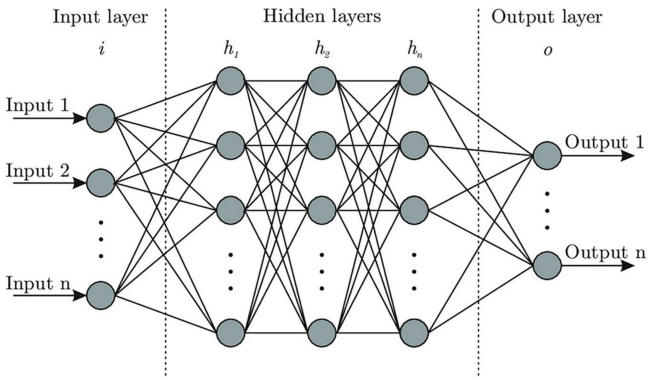


Figure 3.2: An illustration of a neural network, highlighting the input and output layers, and showing the number of middle hidden layers.

minimize a loss function that quantifies the prediction errors.[14] Firstly, let us define the loss function. The loss function, denoted as $L(W)$, quantifies the discrepancy between the predicted outputs of the neural network and the true target values. The primary objective is to minimize this loss by adjusting the parameters W. The choice of a suitable loss function depends on the nature of the task and the type of data being used. Commonly used loss functions include:

- Mean Squared Error (MSE): This is a commonly used loss function for regression tasks. It measures the average of the squared differences between predicted and actual values. The MSE loss for a dataset with N samples is defined as:

$$L(W) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

  where y_i represents the actual target value, $\hat{y}_i$ is the predicted value, and N is the number of samples.

- Cross-Entropy Loss (Log Loss): This loss function is commonly used for classification tasks. It quantifies the dissimilarity between predicted

class probabilities and true labels. For binary classification, the cross-entropy loss is defined as:

$$L(W) = -\frac{1}{N}\sum_{i=1}^{N}(y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i))$$

where y_i is the true label (0 or 1), $\hat{y}_i$ is the predicted probability, and N is the number of samples.

- Categorical Cross-Entropy: This is an extension of cross-entropy for multi-class classification. It measures the dissimilarity between predicted class probabilities and one-hot encoded true labels.

The loss function serves as the guiding criterion for the optimization process. The objective is to find the set of parameters W that minimizes this loss. We call training loss the measure of the error or discrepancy between the predicted output of a neural network and the actual target values on the training dataset i.e. the value of the chosen loss function.

An optimization algorithm is used to update the network's parameters to minimize the loss function. Optimization algorithms that use only the gradient, such as gradient descent, are called first-order optimization algorithms. Optimization algorithms that also use the Hessian matrix, such as Newton's method, are called second-order optimization algorithms. Mathematically, the aim is to adjust the parameters in the direction that decreases the loss. The weight update at each iteration is defined as:

$$W_{t+1} = W_t - \varepsilon \nabla L(W_t)$$

Here, W_{t+1} represents the updated parameters, W_t is the current set of weights, and ε denotes the learning rate, which controls the step size for parameter updates. The backpropagation algorithm is used to compute the gradient of the loss function applied to the output of the network with respect to the parameters in each layer. This gradient can then be passed to the gradient-based optimization algorithm. Various optimization algorithm

variants have been developed to enhance training efficiency and address challenges such as slow convergence and local minima. These include Stochastic Gradient Descent (SGD), Momentum, Adaptive Learning Rate methods like Adagrad, RMSprop, and Adam, as well as Nesterov Accelerated Gradient (NAG). Selecting an appropriate optimization algorithm and fine-tuning hyperparameters is a crucial aspect of training neural networks, as it directly affects the model's convergence and performance.

In traditional optimization algorithms like standard gradient descent, a fixed learning rate ε is used throughout the training process. While this approach can work well for some problems, it may lead to slow convergence or even divergence for others. Adaptive learning rate algorithms address this issue by dynamically adjusting the learning rate for each parameter based on the past behavior of the optimization process.

The adaptive learning rate is typically determined by considering the gradient information, the history of parameter updates, or a combination of both. These algorithms aim to provide a balance between taking large steps for faster convergence and small steps for stable optimization.

The commonly used adaptive learning rate algorithms are[14]:

- Adagrad (Adaptive Gradient Descent): Adagrad adapts the learning rate for each parameter based on the magnitude of its historical gradients. Parameters that have received large gradients in the past will have their learning rate reduced, while those with small gradients will receive larger updates.

- RMSprop (Root Mean Square Propagation): RMSprop is similar to Adagrad but uses a moving average of the squared gradients to adapt the learning rate. It helps mitigate the diminishing learning rate problem encountered in Adagrad.

- Adam (Adaptive Moment Estimation): Adam combines the benefits of both momentum and RMSprop. It maintains moving averages of both gradients and squared gradients and adapts the learning rate for each

parameter. Adam is known for its robust performance in various tasks.

These adaptive learning rate algorithms offer different strategies for dynamically adjusting the learning rate, making them well-suited for a variety of optimization scenarios. The choice of an algorithm depends on the specific problem and the neural network architecture. Understanding their characteristics and behaviors is essential for effective training. In this context, an epoch refers to a single pass through the entire training dataset. It consists of presenting each training sample once to the model, computing the gradients, and updating the weights of the network. What we actually are interested in is not the training loss per se, since a very overfitted model, which is actually 'learning by memory' all the data that it is fed, has an extremely low training loss. We want to measure the generalization capabilities of the network hence we study the validation loss. The validation loss is a metric used to assess the performance of a neural network by using a separate validation dataset. It quantifies the error or loss of the model on data not used during training. A lower validation loss indicates better generalization, suggesting that the model can make accurate predictions on new, unseen data.

3.2.1 Overfitting

Overfitting represents one of the main challenges in machine learning, and understanding its dynamics would be crucial for model evaluation. Overfitting occurs when a model learns the training data too well, capturing noise and fluctuations in addition to the underlying patterns. As a result, an overfitted model may perform exceptionally well on the training set but fails to generalize to new, unseen data.

The phenomenon of overfitting can be illustrated graphically by comparing the training and validation loss over epochs. The validation loss is the loss calculated on a separate validation dataset. In an ideal scenario, as the model is trained, both the training and validation losses should decrease. However, overfitting becomes apparent when the training loss continues to decrease, while the validation loss starts to increase, as can be seen in figure 3.3. In

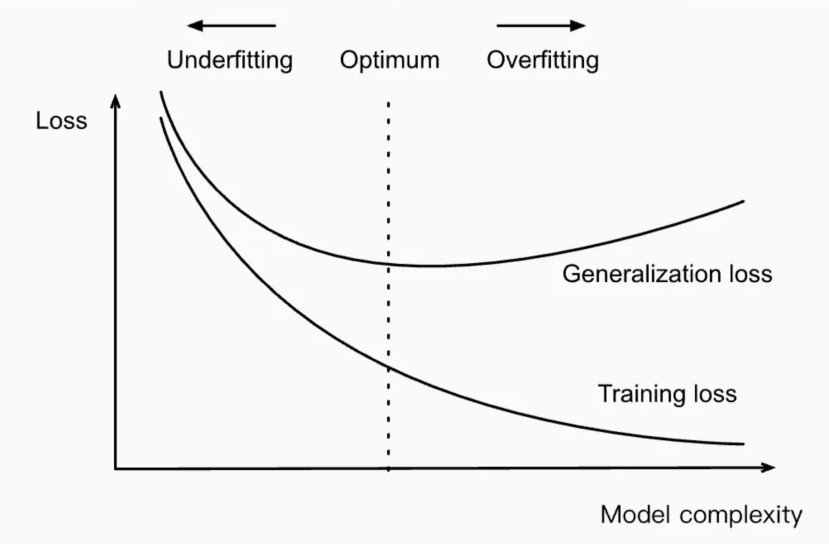


Figure 3.3: Illustration of overfitting in a neural network.

the early epochs, both training and validation losses decrease, indicating that the model is learning and improving its performance. However, as training progresses, the training loss continues to decrease, approaching zero, while the validation loss begins to rise. This divergence suggests that the model is starting to memorize only the training data losing the ability to generalize to the new data. To mitigate overfitting, various techniques are employed during the training process: Early Stopping Monitoring refers to the use of evaluating the validation loss during training and stopping the training process when the validation loss stops improving. The main drawback is that in order to use this approach we need to keep a significant amount of data that the model cannot see and cannot learn from. Also appropriately choosing the networks architecture can be very useful in mitigating overfitting, although there are best practices to do so there are not yet general techniques that tell us what is the best architecture to choose and why.

Chapter 4

State of the art

As we have already seen in chapter 1, the application of TDA in the context of machine learning has been rapidly growing. Since our interest is to analyze the evolution of the weighted graph structure of the network, and how it correlates to the validation loss, let us see the state of the art in this specific research avenue. The use of persistent homology, in particular Betti numbers applied to a suited function space, has also been theoretically proven to be of use for expressing a network's capacity, in particular for comparing Deep and Shallow neural networks [3]. The work in [32] begins by constructing a topological space based on the network's output (actually, on the coverings of the boundary of the output set), and then studies the Betti numbers associated to the Mayer-Vietoris construction built on that space. The authors provide theoretical bounds for the Betti numbers, and apply their results in the context of architecture selection rather than validation error estimation. In [4] a new dimension measure called 'persistent homology dimension' is created, and proven to be equivalent to the box dimension which in turn has been proven to be related to the network's generalization capabilities [35]. In [7] two types of path homology for fully-connected, feed-forward neural network architectures is provided, and shown to be able to detect some information about the network's initialization. Path homology, as the name suggests, is defined on a chain of groups whose elements are pos-

sible paths on the complex. Although technically neural networks do have a structure of directed graphs, due to the difference in meaning attributed to positive/negative weights, we have decided to discard the directional information from a path-homology point of view. Moreover, we have found two papers that have an approach that is most similar to ours, and let us go into detail about their method and their findings in the next sections. Each of these papers is focused on analyzing the graph structure of the network in order to assess its generalizing capabilities.

4.1 Neural Persistence

In the paper [26] the proposed study is done for feedforward neural networks, which can be seen as a stratified graph, where the authors denote with W the set of weights. Since W is typically changing during training, at each training step the function $\phi : E \to W$ that maps edges to weights is required. If W is the set of weights for the current training step, $w_{\max} := \max_{w \in W} |w|$. Then its defined $W_0 := \left\{ \frac{|w|}{w_{\max}} \mid w \in W \right\}$, the set of transformed weights in non-ascending order, such that $1 = w_{0,0} \geq w_{0,1} \geq \ldots \geq 0$. This allows the definition of a filtration for the k-th layer using the transformed weights of the edges. The filtration ensures network invariance to scaling ($w_0 \in [0, 1]$), simplifying cross-network comparisons. Then, the authors calculate the 0-dimensional persistent homology groups for every layer. The focus on 0-dimensional information is due to it being more efficiently computable and easily interpretable. Then, the authors define a scalar value associated to the diagram as

Definition 24. *The neural persistence associated to the k-th layer G_k is*

$$NP(G_k) := ||D_k||_p = \left(\sum_{(c,d) \in D_k} pers(c, d)^p \right)^{\frac{1}{p}}, \quad (1)$$

which, for $p = 2$, captures the Euclidean distance of points in D_k to the diagonal.

Then, the authors prove the following upper bound:

Theorem 1. *Let G_k be a layer of a neural network according to Definition 1. Furthermore, let $\phi_k : E_k \to W_0$ denote the function that assigns each edge of G_k a transformed weight. Using the filtration from Section 3.1 to calculate persistent homology, the neural persistence $NP(G_k)$ of the k-th layer satisfies*

$$0 \leq NP(G_k) \leq \left(\sum_{e \in E_k} \phi_k(e) - \min_{e \in E_k} \phi_k(e) \right)^{\frac{1}{p}} (|V_k \times V_{k+1}| - 1)^{\frac{1}{p}}, \quad (2)$$

where $|V_k \times V_{k+1}|$ denotes the cardinality of the vertex set, i.e., the number of neurons in the layer.

which then is used to normalize the neural persistence of a layer, making it possible to compare layers (and neural networks) that feature different neural combinations.

Definition 25. *(Normalized neural persistence). For a layer G_k following Definition 1, using the upper bound of Theorem 1, the normalized neural persistence $NP(\mathcal{G}_k)$ is defined as the neural persistence of G_k divided by its upper bound, i.e.,*

$$NP(\mathcal{G}_k) := \frac{NP(G_k)}{NP(G_k^+)}.$$

Finally, the give the following definition, obtaining a parameter on the whole graph.

Definition 26. *(Mean normalized neural persistence). Considering a network as a stratified graph G, the sum of the neural persistence values per layer is the mean normalized neural persistence, i.e., $NP(G) := \frac{1}{l} \sum_{k=0}^{l-1} NP(f(G_k))$.*

Finally, the authors prove upper and lower bounds for the mean normalized neural persistence. The first experimental result is shown in figure 4.1. Then, the authors use the mean normalized neural persistence as an early stopping criterion, and show their results obtained on FashionMNIST, MNIST, Cifar10 and IMDB, which we can see in 4.2.

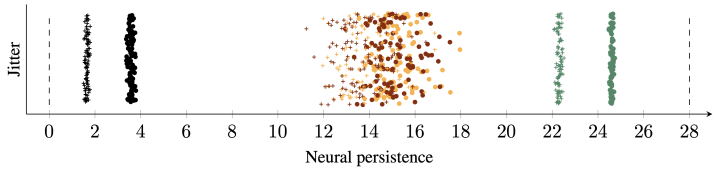


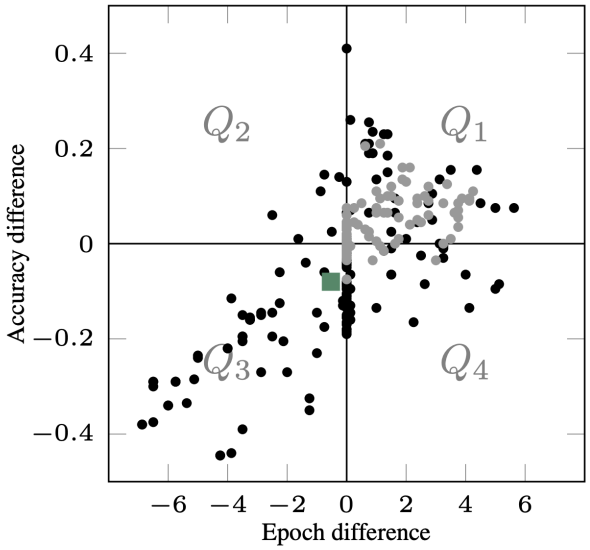
Figure 4.1: Neural persistence values of trained perceptrons (green), diverging ones (yellow), random Gaussian matrices (red), and random uniform matrices (black). The authors performed 100 runs per category and dots indicate neural persistence while crosses indicate the predicted lower bound according to their theorem. The upper bounds are not shown as not strict enough to be meaningful.

4.2 Relevance

The other published paper that uses the same tool to the same end is the paper [30] by Watanabe and Yamana. In this paper the positive weights connecting neurons of adjacent layers are normalized according to the following calculation: $R_{ij} = \frac{w_{ij}}{\sum_{i,i\neq j} w_{ij}}$. Then the authors define the relevance between neurons of distant layers as

$$R_{ij} = \max_{(v_i, v_{m_1}, \ldots, v_{m_k}, v_j) \in L_{ij}} R_{v_i v_{m_1}} \cdots R_{v_{m_k} v_j},$$

where L_{ij} denotes the set of all possible paths from v_i to v_j. Hence, the authors associated a fully connected graph to the neural network, with weight function R. Then, the authors consider the weighted simplicial complex associated to the above mentioned graph, and consider the sublevel set filtration. The authors train two neural networks, one on the MNIST dataset, the other on the CIFAR10 dataset. Then, the authors consider the final weights of the successfully trained network and plot the associated 0 dimensional persistence diagrams, as can be seen in 4.3. The authors claim that the following three observations can be made from the figure: (1) points are plotted in the belt-like area (birth+5 $<$ death $<$ birth+20) parallel to the dialog line; (2) some figures have points below the belt-like area; and (3) some figures have points over the belt-like area. With respect to observation (2), the number



(a) Fashion-MNIST

Data set	Barycentre	Final test accuracy
Fashion-MNIST	$(-0.53, -0.08)$	86.72 ± 0.43
MNIST	$(+0.17, -0.06)$	96.16 ± 0.24
CIFAR-10	$(-1.33, -1.13)$	52.19 ± 3.40
IMDB	$(-1.68, +0.07)$	87.35 ± 0.03

(b) Summary

Figure 4.2: Figure (*b*) depict the differences in accuracy and epoch for all comparison scenarios of mean normalized neural persistence versus validation loss, in the case of the FashionMNIST dataset, while the table summarizes the results obtained on the other data sets.

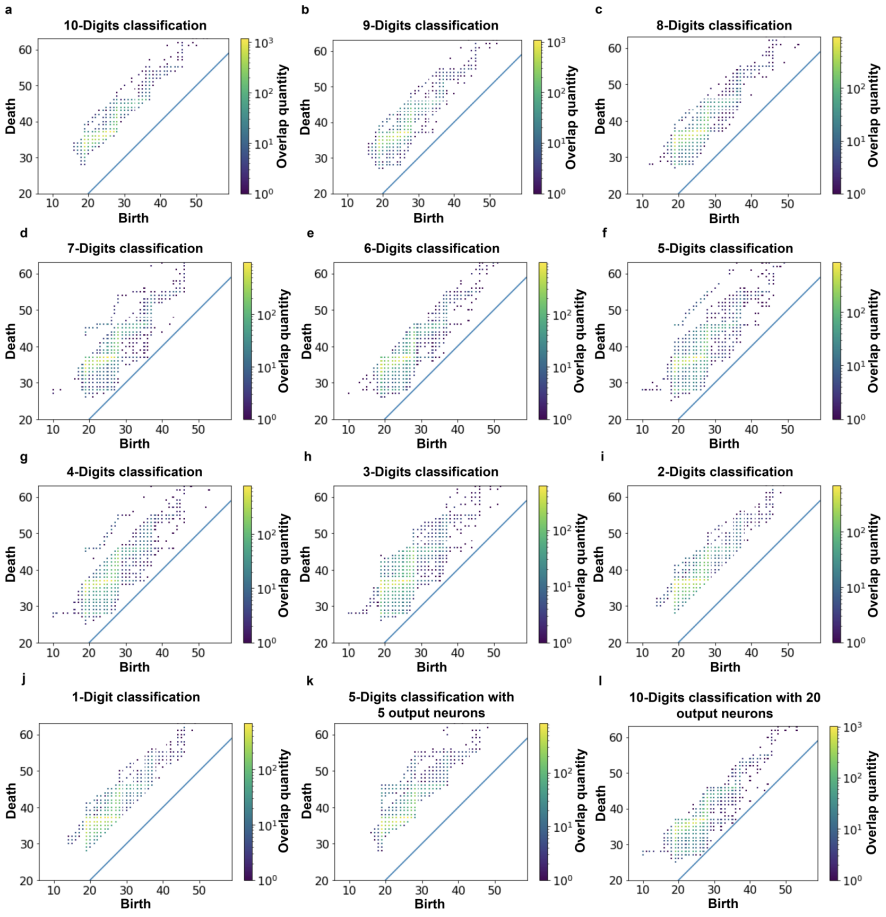


Figure 4.3: Persistence diagrams, obtained for a neural network trained on the MNIST dataset, as some parameters of the network are varied.

of points below the belt-like area increases from figure (a) to figure (g) and decreases from figure (h) to figure (j). The authors claim that this pattern reflects both the excess of the output neurons and problem difficulty. It can be further observed that the diagrams seem to reflect the degree of confidence of the models, i.e., the excess of the output neurons reduced the confidence, whereas the simplicity of the problem increased it.

Chapter 5

Our Contribution

After having introduced some notions, necessary in our modelization, in chapter 3 we are now ready to use the concepts we defined to our aim. Our goal was to propose a sound theoretical setting that can be used to study the connections between the topological properties of the graph associated to a neural network. Then, we will use the model we propose in order to find a connection between the training step, hence the evolution of the weights, and the validation loss. Finding such correlation would provide firstly a tool that can be used in conditions of data scarcity since there is no need to withhold a portion of the dataset, moreover it could be a possible approach in shedding light on why neural networks possess such great and quick generalization capabilities.

5.1 Pipeline

Let us now show the steps taken in our process. We have evaluated two metrics, one which is relative to the global graph structure of the neural network, the other to the local (layerwise) graph structure of the network. The general approach is the following: a weighted graph is associated to the neural network, from this weighted graph a weighted abstract simplicial complex is defined. We consider the sublevel set filtration defined in Chapter

2. Then, the persistent homology groups associated to this filtration are calculated and their features represented in persistence diagrams. Finally, a scalar value is associated to these diagrams, which we will use as a proxy for the validation loss.
Firstly, let us define the graph associated to a feedforward neural network, $G = (V, E)$. The connections within the network can be conceptualized as forming a stratified graph.

Definition 27. *A stratified graph is a multipartite graph $G = (V, E)$ such that $V = V_0 \cup V_1 \cup ...$, implying that if $u \in V_i$, $v \in V_j$, and $(u, v) \in E$, then $j = i + 1$. Hence, edges are exclusively allowed between adjacent vertex sets. For a given positive integer $k \in \mathbb{N}$, the k-th layer of a stratified graph is defined as the unique subgraph $(G_k := (V_k \cup V_{k+1}, E_k := E \cap \{V_k \times V_{k+1}\})$.*

Hence, the following steps will be taken at each iteration of the training process. Since to each edge, during training, is associated a weight, we can consider the weight function $w(e) : E \to \mathbb{R}$ which sometimes will be noted as $w(u, v)$ for $e = (u, v)$. Notice how at each epoch there will be a different weight function, although we have omitted this in the notation for simplification. Let w be the weight function at the current training step. For the global method we either use directly the associated graph, or define a graph that spans across layers, and by showing two types of normalizations of the weights we get two possible filtrations. For the layer method we associate a graph, and hence a simplicial complex, to each layer, and for each of those we compute our metric, which we then average at the end. Also for the layer method we have devised two possible normalization techniques. For each developed filtration the persistent homology groups are calculated and then, for dimensions 1 in the global case and dimension 0 for the layer case, the persistence diagrams are calculated. For each diagram, we then evaluate our scalar parameter. Since this process is repeated for every step of the training procedure, we will obtain a history of the parameter as we advance in epochs. Let us now show specifically how the process differs for the global and layer method.

5.1.1 Local normalizaton

The local method consists in evaluating the persistent homology groups only relative to each layer subgraph. This layerwise approach was inspired by paper [26]. Let us show this procedure for the generic k-th layer. Let $G_k = (V_k \cup V_{k+1}, E_k)$ be the k-th subgraph.

Now, in order to define a filtration, we use the weights associated to the network's edges. For this local method we have evaluated and decided to compare two different normalization techniques on the weights:

- Probability Normalization: Each layers weights are taken in absolute value and divided by the sum of the absolute value of the weights that have the same target neuron. Formally:

$$w'(u,v) := \frac{|w(u,v)|}{\sum_{p \in V_k} |w(p,v)|}$$

  for $(u,v) \in E_k$.

- Layerwise Normalization: Each layers weights are taken in absolute value and divided by the greatest weight of that layer in absolute value. Formally:

$$w'(e_k) := \frac{|w(e_k)|}{\max_{s_k \in E_k} |w(s_k)|}$$

  for $e_k \in E_k$.

We can see a simple example of these normalizations in figure 5.1. Hence we will be considering the graphs G_k with weights w'.

5.1.2 Global normalization

For the global method the approach is quite different. Consider the graph associated to the network, and let w denote the weight function on the whole graph, which is defined only on couples of vertices of adjacent layers. We again compare two different normalization techniques on the weights:

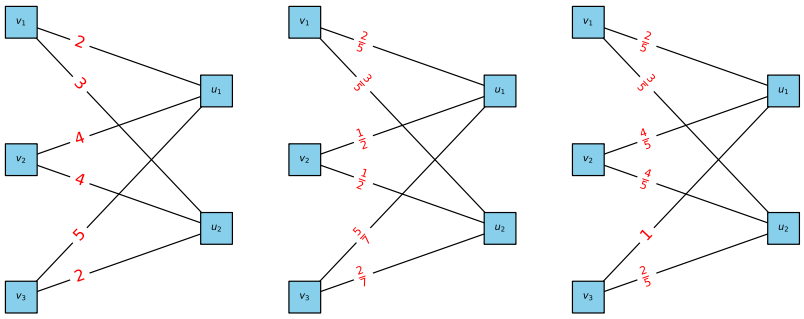


Figure 5.1: On the left we can see a single layer, in the middle the same layer normalized using the Probability Normalization and on the right the Layerwise Normalization.

- Global Normalization: Each layers weights are taken in absolute value and divided by the greatest weight amongst all layers, in absolute value. Formally:

$$w'(e_k) := \frac{|w(e_k)|}{\max_{s \in E} |w(s)|}$$

for $e_k \in E_k$.

- Probability Normalization: Each layers weights are taken in absolute value and divided by the sum of the absolute value of the weights that have the same target neuron. Formally:

$$w'(u, v) := \frac{|w(u, v)|}{\sum_{p \in V_k} |w(p, v)|}$$

for $(u, v) \in E_k$.

We can see a simple example of these normalizations in figure 5.2. Now, we can take two different approaches. We can simply consider the graph associated to the neural network with normalized weight function w'. The other approach is to define a new weight function, starting from the normalized weights w' called relevance and defined on all $V \times V$, similarly as what was done in [30]:

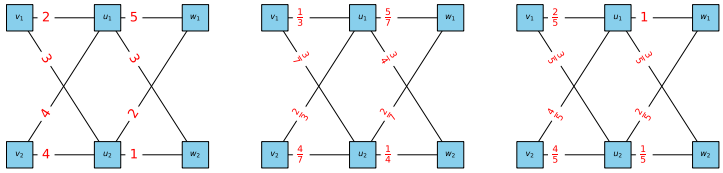


Figure 5.2: On the left we can see a simple graph composed of two layers, in the middle the same graph normalized using the Probability Normalization and on the right the graph's weights normalized using the Global Normalization.

Definition 28. *The relevance function is the function* $R : V \times V \to \mathbb{R}$ *such that*

$$R(v,u) = \begin{cases} 0 & \text{if } v,u \in V_i, \\ \max\limits_{y_1 \dots y_s \in \Gamma_{v,u}} w'(v,y_1)\dots w'(y_s,u) & \text{otherwise.} \end{cases}$$

Where $\Gamma_{v,u}$ *denotes the possible paths from* v *to* u*.*

We will call the relevance between v and u the *max product path from* v *to* u. The intuition behind the definition is as follows: $R_{0,1}$ and $R_{1,2}$ indicate the contributions of v_0 and v_1 to the increase in the inputs of v_1 and v_2, respectively, and $R_{0,2}$ indicates the contribution of v_0 to the increase in the input of v_2. Hence, the graph we consider in this case is the graph that has vertices V and edges R. It is important to keep in mind that R clearly depends on the choice of normalization. In order to find an efficient way to calculate this we need to define the following matricial product:

Definition 29. *Let* $A \in Mat_{n\times m}(\mathbb{R})$ *and* $B \in Mat_{m\times p}(\mathbb{R})$*. The grid-max product of* A *and* B*, denoted with* $A \odot_{gmax} B$ *is the* $n \times p$ *matrix defined as:* $(A \odot_{gmax} B)_{ij} = \max\limits_k a_{ik} \cdot b_{kj}$

Now we can prove the following proposition:

Proposition 1. *Let* G_w *be a stratified feedforward weighted graph and let* W^i *the matrix representing the weights between layers* i *and* $i+1$*. Let us denote*

with v_r^i the $r-th$ node of the $i-th$ layer.
Then the max product path from v_r^i to v_s^j is

$$\left(((W^i \odot_{gmax} W^{i+1}) \odot_{gmax} \dots W^{j-1}) \odot_{gmax} W^j\right)_{rs}$$

Proof. Without loss of generality let us assume $i = 0$.
Let us prove this by induction on j: By inductive hypothesis the max product path from v_r^0 to v_k^{j-1} is $c_{rk} := (((W^i \odot_{gmax} W^{i+1}) \odot_{gmax} \dots W^{j-1})_{rk}$ for all $v_k^{j-1} \in V_{j-1}$. Then, the possible paths products from v_r^i to v_s^j are $c_{rk} \cdot W_{ks}^j$ as k varies. The maximum, as k varies, yields the max product path from v_r^i to v_s^j. This is exactly the grid-max product $(C \odot_{gmax} W^j)_{rs}$. □

This theorem allows for a very efficient computation of the relevance matrix, whose evaluation would otherwise be computationally very expensive. The final relevance matrix R defined before, which we can compute by iteratively applying the grid-max products, represents the weight function for our new graph.

5.2 Filtration

Now, we have to associate a filtration and a simplicial complex to the considered weighted graph. Hence, for each choice of normalization, we have either a sequence of weighted graphs $(G_1, w_1')...(G_n, w_n')$ for the layer by layer scenario or a weighted graph (G, w') for the global case. As we have defined in chapter 2, for each weighted graph, we can consider the associated weighted simplicial complex $(K(G), \bar{w'})$.

The filtration we will be considering is the sublevel set filtration, as previously defined, that we also show here as a reminder.

Definition 30. *Given a weighted simplicial complex, with monotone weights (K, w) and $w \in [0, 1]$ then, given $0 = w_1 \leq \dots \leq w_n = 1$ the sublevel set filtration is $K_i := \{\sigma \in K | w(\sigma) \leq w_i\}$ and $K_0 := \emptyset$.*

We can actually expand this definition in the following sense:

Definition 31. *Given a weighted simplicial complex, with monotone weights* (K, w) *then, given* $w_1 \leq \ldots \leq w_n = \max w$ *we set* $K_i := \{\sigma \in K | w(\sigma) \leq w_i\}$ *and* $K_0 := \emptyset$*. This is called the expanded sublevel set filtration.*

Since w is monotone, this still induces a chain of inclusions up to K. Recall that, since we are comparing two different types of normalization techniques, they give rise to different filtrations and hence persistent homology groups. Due to its algebraic combinatorial nature, persistent homology groups are very computationally expensive. Only the computation of the 0-th dimensional diagrams is currently very efficient, and already at dimension 1 the computation becomes more expensive[24]. Currently, the state of the art in terms of computing the persistence groups, is restricted to the computation of the Vietoris-Rips homology. That is why, in order to make our computations feasible, we need the following result.

Definition 32. *Given a finite set* X *and a symmetric function* $f : X \times X \to \bar{\mathbb{R}}$*. Then the abstract Vietoris Rips complex at scale* $r \in \bar{\mathbb{R}}$ *is:*

$$VR_r(X) = \{\sigma \subseteq P(X) \mid \forall a, b \in \sigma : f(a, b) \leq r\}$$

Since each $VR_r(X)$ clearly respects the closure under inclusion and intersection, it actually is an abstract simplicial complex.

Definition 33. *Given a finite set* X *and a function* $f : X \times X \to \bar{\mathbb{R}}$*. Let* $Im(f) = \{w_1 \leq \cdots \leq w_N\}$*. Then the abstract Vietoris Rips complex filtration is the filtration given by:*

$$\emptyset \subseteq VR_0(X) \subset VR_{w_1}(X) \subset \cdots \subset VR_{w_N}(X)$$

This is a filtration of the complex $\mathrm{VR}_{w_N}(X)$.
Finally, we can prove the following result:

Proposition 2. *Let* (K, w) *be a weighted finite abstract simplicial 1-complex with monotone weights and* $w_m := \max w$*. Let* $X := S_0(K)$ *be its 0-simplexes*

and $f : S_0 \times S_0 \to \bar{\mathbb{R}}$ *defined by* $f(\sigma_i, \sigma_j) := \begin{cases} +\infty \text{ if } \{\sigma_i, \sigma_j\} \notin S_1(K) \\ w(\{\sigma_i, \sigma_j\}) \text{ if } \{\sigma_i, \sigma_j\} \in S_1(K) \\ w(\sigma_i) \text{ if } \sigma_i = \sigma_j \end{cases}$

Then the expanded sublevel set filtration associated to $Im(w)$ *is isomorphic to the Vietoris-Rips filtration of* $VR_{w_m}(X)$.

Proof. Let $Im(w) = \{w_0 \leq \cdots \leq w_m\}$. Hence $\mathrm{VR}_{w_i}(X) =$
$= \{\sigma \subseteq P(X) \mid \forall a, b \in \sigma : f(a, b) \leq w_i\} =$
$= \{\sigma \in K | w(\sigma) \leq w_i\} = K_{w_i}$. □

This theorem allows us to evaluate our persistent homology groups, hence the diagrams, by using the same approach, and the same tools, as when calculating the Vietoris Rips homology groups. From a computational point of view algorithms developed for the calculation of Vietoris Rips persistent homology groups are much more efficient. Thus, after considering the weighted complex associated to the weighted graph, we can calculate the Vietoris Rips homology groups which have the same persistence. Hence, we have efficiently evaluated the homology groups and we can consider the associated persistence diagrams. Formally:

Definition 34. *Let* D_k *be the k-th-dimensional persistence diagram associated to the homology groups of dimension k. If* $\delta := \max_{(b,d) \in D_k} d$ *we consider the bounded k-th dimensional persistence diagram as* $D_k^* := \{(b, d) \in D_k | d \neq +\infty\} \cup \{(b, \delta) | (b, +\infty) \in D_k\}$.
Then, let us define the k-th norm persistence as

$$\|D_k^*\|_2 := \left(\sum_{(b,d) \in D_k^*} (d - b)^2 \right)^{1/2}$$

Since we are obtaining a scalar value for each layer, when considering the layerwise method, we then average these results in order to get a single parameter for the whole graph. This 2-norm represents a sort of mean of the pair persistences. A higher value corresponds to longer persistences,

meaning multiple classes being born and not merging for most time. The use of p-Norm has already been seen in [26] and [9].

Chapter 6

Experiments

In this section, we show the experimental results obtained comparing methods that try to estimate the validation loss by extracting topological information of the neural network. Using the theoretical framework we constructed we are able to systematically compare techniques and results, starting from what was done in the papers cited in chapter 5. A graph illustrating the steps that we take can be seen in figure 6.1 From a computational standpoint the most complex computation is the one of the homology groups, and hence the persistence diagrams. Luckily, the state of the art in terms of persistent homology is represented by the library Ripser [2], which is the fastest library for the computation of persistent homology [24]. However, this library only calculates the Vietoris-Rips filtration (and then persistent homology), but by using the proposition proven in the previous chapter, we can still use this tool to calculate the persistence diagrams. Our aim is to compare the results obtained as the following parameters are varied:

- Neural Network's architecture (Deep or Wide hidden layers).
- The chosen normalization.
- Layerwise, Global construction.
- For the global construction the use of relevance or just considering the network with its normalized weights.

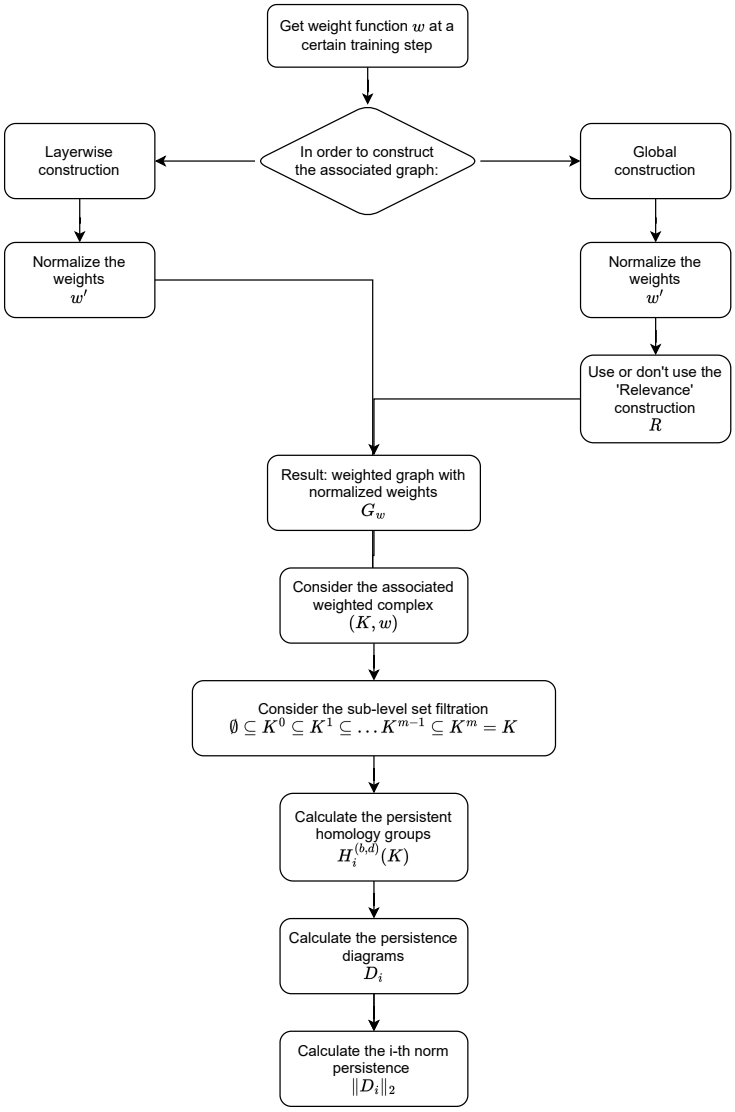


Figure 6.1: The steps taken in our process.

- The training dataset.

Using this general approach, results as seen in the papers cited in chapter 5 are retrieved, computed much more efficiently, and able to be compared to other possible choices of complex construction. For instance, the history of the 0-th norm persistence, with the layerwise normalization, coincides with neural persistence. Moreover, the information about the belt-area identified in the second paper, parallel to the y=x line, could retrieved by the 1-th norm persistence. That model is parameterized in our context by taking the probabilistic normalization, ordering the weights in descending order and using the relevance weight function.

6.0.1 Model Architectures and datasets

The datasets we used when training our network are MNIST, Fashion-MNIST and Cifar10: The MNIST dataset is a collection of 28x28 grayscale images of handwritten digits (0 to 9) where each image is labeled with the corresponding digit. FashionMNIST is comprised 28x28 grayscale images of fashion items such as clothing, shoes, and accessories which offers a more challenging and diverse set of objects for image classification tasks. Like MNIST, each image is labeled with the corresponding fashion category. CIFAR-10, on the other hand, is a dataset that consists of 32x32 color images across ten different classes, including various animals, vehicles, and everyday objects. CIFAR-10 provides the most complex and realistic set of images compared to MNIST and FashionMNIST. Regarding the chosen architectures, we decided to employ neural network architectures with variations in depth and width. For the wide neural networks, we explore architectures with a single hidden layer, where the width is varied in increments of 200. Additionally, for deep neural networks, architectures with 2, 3, and 4 hidden layers, each with 100 units, are investigated. Deeper and Wider networks become computationally expensive, in the case of the relevance-graph calculation, because of the combinatorial explosion of edges in the complex associated to it. We have always used the optimizer Adam with learning rate $1e-3$, as SGD needed

too many epochs to be brought to overfitting thus making it not feasible to evaluate our metrics for so many tries. We have used Cross Entropy as the loss function. We have trained the network for 100 epochs for each experiment and in order to keep the computational cost not too heavy we have evaluated our metric at each epoch for the first 40 epochs, then once every 10 epochs.

6.0.2 Expected and observed behaviours

In order to extract relevant information about the network we have used the following technique: in order to define the filtration, since after normalization weights are sorted from the lowest to the highest, we have considered as weight function the function $1 - w'$. This choice, which is the one also usually adopted in the literature, is due to what phenomenon we are trying to enhance; since large weights (after normalization) mean that the flow of information passing through that edge seems quite relevant, we want to associate it a high persistence. In order to do so, we would want it to appear early and "die" after quite some time in the filtration process. Hence why we will be considering the graph $G_w := (V, E, w := 1 - w')$ at each training step. When using the layerwise method we will be considering only the 0-th norm persistence as the 1 dimensional components (holes) created in the layerwise graph do not carry any information about the network's flow of information. A stable and high 0 dimensional persistence is associated with sparsity of the associated graph hence an efficient use by the network of the information coming from the previous layer, at each layer. A lower persistence would indicate that the network probably stopped generalizing and is constantly readjusting its weights in order to memorize the single data and thus overfitting. On the other hand, when considering the global network, whether using the relevance and considering connections across layers or just considering the whole graph with its normalized weights, since the birth of a new edge on a layer would kill an homology class in the adjacent layers we postulate that this would not be an interesting information to gather from

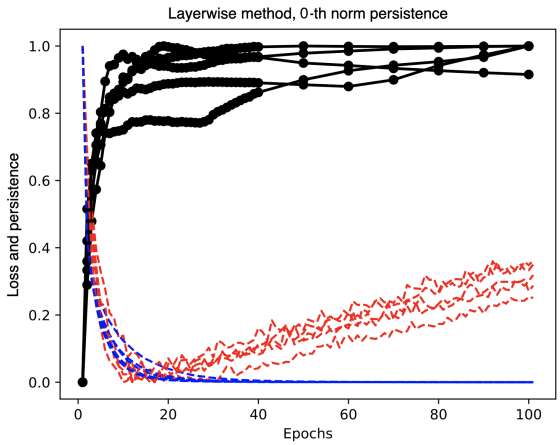


Figure 6.2: For the layerwise method we have computed the 0-th norm persistence which can be seen in black, validation loss in red and training loss in blue. All lines have been normalized in order to have values between 0 and 1 in order to better illustrate their trend. We have shown the results from all 5 experiments on the same figure in order to show the common behaviour.

the graph. In these cases we will be considering the 1-dimensional norm persistence. Now, the presence of highly persistent classes would indicate that the information is travelling from the same nodes to the same target node by different routes. Hence, we would correlate *low* 1-dimensional norm persistence with an efficient flow of information. Let us now see a first set of experiments in order to validate these hypotheses. For each figure we have trained 5 neural networks on the MNIST dataset, using architectures with hidden layers of dimensions 200, 400, 600, 800 and 1000. We have plotted on the same figure the 0-th norm persistence, the validation loss and the training loss. In figure 6.2 we can see the 0-th norm persistence for these 5 executions. As we can see this metric grows and seems to reach a plateau. Next, we have compared, using the same setting, the 0-th norm persistence with the 1-th norm persistence associated to the global method. The results of these experiments can be seen in figure 6.3. As we predicted the 0-th norm persistence doesn't seem to carry any relevant information about the training procedure. The same comparison has been evaluated for the method where

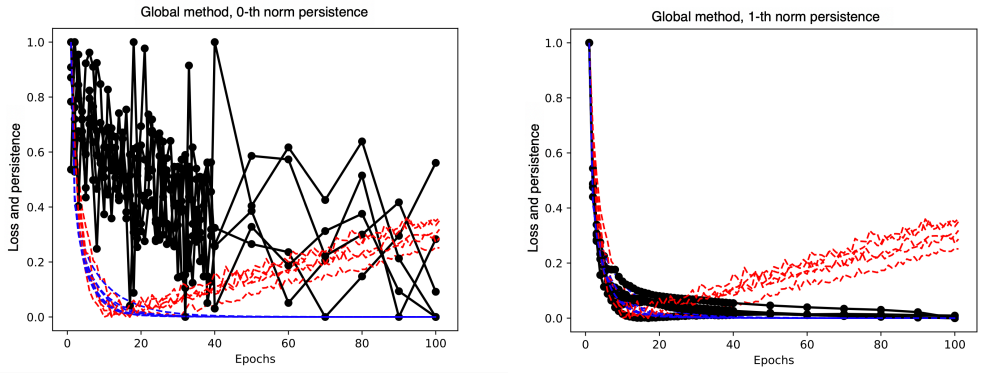


Figure 6.3: For the global method we have here computed the 0-th norm persistence on the left and the 1-th norm persistence on the right which can be seen in black. Validation loss is in red and training loss in blue. All lines have been normalized in order to have values between 0 and 1.

the graph is constructed using the relevance weight function and the results are displayed in figures 6.4.

6.0.3 Early stopping

The main advantage of a proxy for the validation loss would be an early stopping criterion: since our objective is to stop the training process when the network has learned the underlying logic behind the data that it has been training on, we want to stop when its generalizing capabilities are at maximum and not advance the training. Advancing in the training procedure would lead to overfitting, and that is precisely the problem that we want to address. Since we are striving for high and stable 0-th norm persistence we stop at the first local maximum that is found using a patience parameter that waits 10 epochs in order to determine if we actually are at a local maximum. On the other hand since we want low 1-dimensional persistence, for the global method and the method that uses the relevance construction we stop at the first local minimum waiting with a patience of 10 epochs. Firstly we can see the results obtained using the Probabilistic normalization for all models

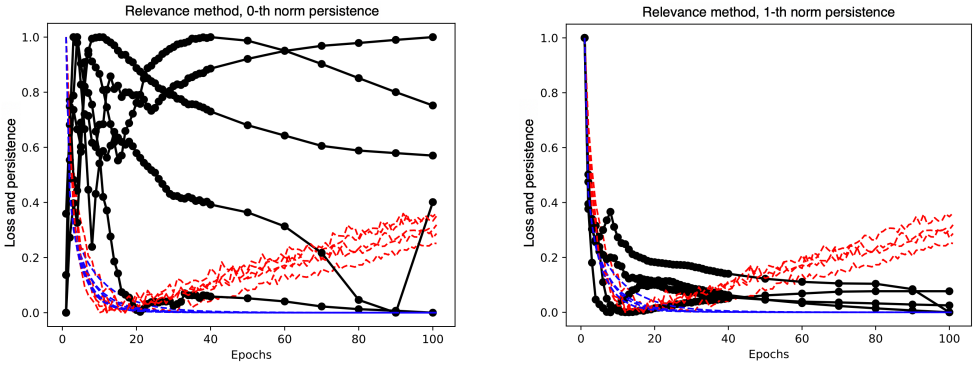


Figure 6.4: For the relevance method we have here computed the 0-th norm persistence on the left and the 1-th norm persistence on the right which can be seen in black. Validation loss is in red and training loss in blue. All lines have been normalized in order to have values between 0 and 1.

in figure 6.5. On the x axis we have the difference of the epoch where the validation loss actually reaches the minimum and the epoch at which the early stopping is triggered. On the y axis we show the following parameter:

$$\frac{(\text{validation loss at stopping point}) - (\text{lowest validation loss})}{(\text{validation loss at last epoch}) - (\text{lowest validation loss})}$$

this parameter gives us a value between 0 and 1 where 0 indicates that we stopped at the exact minimum whereas 1 means that the early stopping criterion was never triggered. As we can see from figure 6.5 this normalization seems better suited for the global method, rather than the layerwise method which seems to be triggered later than optimal. Now let us compare the results obtained with the Layerwise normalization for the layerwise method and the corresponding global normalization used for the global and relevance methods. As we can see from figure 6.6 this kind of normalization seems better suited for the layerwise method than the global methods. The layerwise method, with the layerwise normalization, is very close to the parameter called "neural persistence" proposed in [26]. As we can see by comparing these two figures the global method using the probability normalization achieves slightly better results. By applying our standardized method

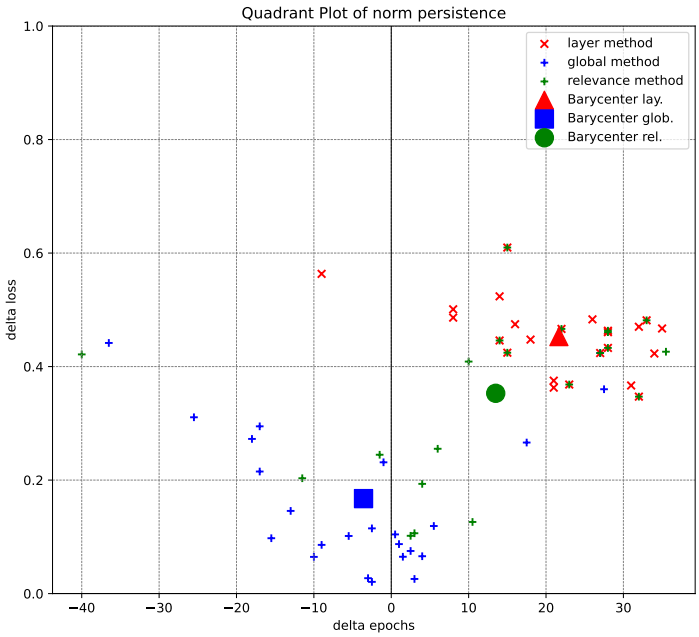


Figure 6.5: In this figure we have mapped a point for each simulation that was executed. The network's architecture was varied, by widening the hidden layer and deepening the structure. Simulations have been made using the datasets MNIST, FashionMNIST and Cifar10. In red we can see the results for the layerwise method, in green for the global method and in blue for the relevance method.

we hence were able to efficiently compare different results.

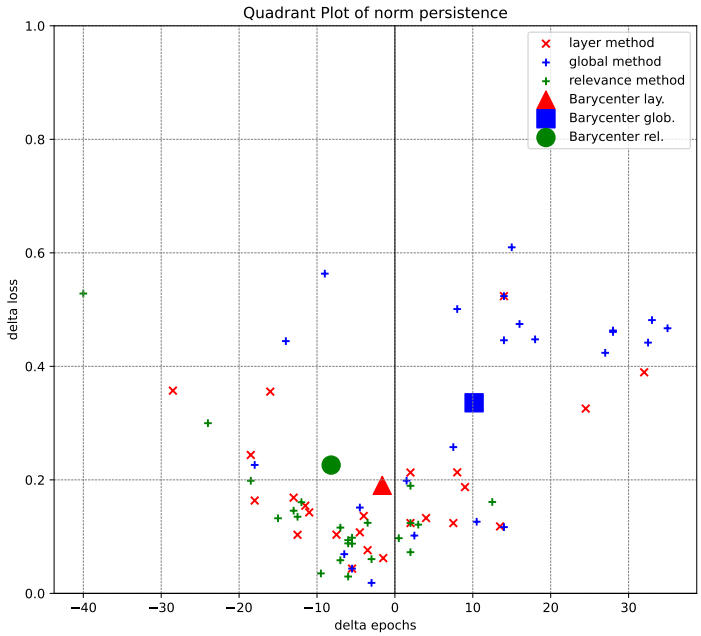


Figure 6.6: In red we can see the results for the layerwise method, in green for the global method and in blue for the relevance method.

Conclusions

In this thesis we have developed a formal standardized framework that allows the efficient computation of the persistent homology associated to a feedforward stratified graph. By leveraging state of the art techniques in the field of topological machine learning and introducing some novel notions related to weighted simplicial complexes, we have proved some theoretical results that allow the use of the most efficient tools present for the concrete application of these theories. Moreover, we tackled the issue of estimating the validation loss in data scarcity scenarios. By developing an early stopping criterion based on topological properties of the network, we are able to prevent overfitting without withholding a portion of the training dataset, which would be particularly useful in data scarcity scenarios. Some interesting avenues of research could be focused on finding optimal normalization techniques, particularly ones that address the different subtle differences associated to positive and negative weights.

Bibliography

[1] Mehmet E Aktas, Esra Akbas, and Ahmed El Fatmaoui. "Persistence homology of networks: methods and applications". In: *Applied Network Science* 4.1 (2019), pp. 1–28.

[2] Ulrich Bauer. "Ripser: efficient computation of Vietoris–Rips persistence barcodes". In: *Journal of Applied and Computational Topology* 5.3 (2021), pp. 391–423.

[3] Monica Bianchini and Franco Scarselli. "On the complexity of shallow and deep neural network classifiers". In: *The European Symposium on Artificial Neural Networks*. 2014. URL: https://api.semanticscholar.org/CorpusID:471567.

[4] Tolga Birdal et al. *Intrinsic Dimension, Persistent Homology and Generalization in Neural Networks*. 2021. arXiv: 2111.13171 [cs.LG].

[5] Mathieu Carrière et al. "Perslay: A neural network layer for persistence diagrams and new graph topological signatures". In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2020, pp. 2786–2796.

[6] Chao Chen et al. "A topological regularizer for classifiers via persistent homology". In: *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR. 2019, pp. 2573–2582.

[7] Samir Chowdhury et al. "Path Homologies of Deep Feedforward Networks". In: *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*. IEEE, 2019. DOI: 10.1109/

icmla.2019.00181. URL: https://doi.org/10.1109%2Ficmla.2019.00181.

[8] David Cohen-Steiner, Herbert Edelsbrunner, and John Harer. "Stability of Persistence Diagrams". In: vol. 37. June 2005, pp. 263–271. DOI: 10.1007/s00454-006-1276-5.

[9] David Cohen-Steiner et al. "Lipschitz functions have L p-stable persistence". In: *Foundations of computational mathematics* 10.2 (2010), pp. 127–139.

[10] Edelsbrunner, Letscher, and Zomorodian. "Topological persistence and simplification". In: *Discrete & Computational Geometry* 28 (2002), pp. 511–533.

[11] Herbert Edelsbrunner, Dmitriy Morozov, and Lawrence Berkeley. "Persistent Homology: Theory and Practice". In: 2012. URL: https://api.semanticscholar.org/CorpusID:9906139.

[12] Rickard Brüel Gabrielsson and Gunnar Carlsson. "Exposition and interpretation of the topology of neural networks". In: *2019 18th ieee international conference on machine learning and applications (icmla)*. IEEE. 2019, pp. 1069–1076.

[13] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep Sparse Rectifier Neural Networks". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Geoffrey Gordon, David Dunson, and Miroslav Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, 2011, pp. 315–323. URL: https://proceedings.mlr.press/v15/glorot11a.html.

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

[15] Asier Gutiérrez-Fandiño et al. "Persistent homology captures the generalization of neural networks without a validation set". In: *arXiv preprint arXiv:2106.00012* (2021).

[16] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. 2nd ed. New York, NY: Springer, 2009, pp. XXII, 745. ISBN: 978-0-387-84857-0. DOI: 10.1007/978-0-387-84858-7.

[17] Christoph Hofer et al. "Deep learning with topological signatures". In: *Advances in neural information processing systems* 30 (2017).

[18] Danijela Horak, Slobodan Maletić, and Milan Rajković. "Persistent homology of complex networks". In: *Journal of Statistical Mechanics: Theory and Experiment* 2009.03 (2009), P03034.

[19] Harish Kannan et al. "Persistent homology of unweighted complex networks via discrete Morse theory". In: *Scientific Reports* 9.1 (2019). DOI: 10.1038/s41598-019-50202-3. URL: https://doi.org/10.1038%2Fs41598-019-50202-3.

[20] Kwangho Kim et al. "Pllay: Efficient topological layer based on persistent landscapes". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 15965–15977.

[21] Fabio Marcuzzi. *Numerical Linear Algebra and Learning from Data, note del corso*. Università degli Studi di Padova, 2022.

[22] Michael Moor et al. "Topological autoencoders". In: *International conference on machine learning*. PMLR. 2020, pp. 7045–7054.

[23] Nina Otter et al. "A roadmap for the computation of persistent homology". In: *EPJ Data Science* 6.1 (2017). DOI: 10.1140/epjds/s13688-017-0109-5. URL: https://doi.org/10.1140%2Fepjds%2Fs13688-017-0109-5.

[24] Nina Otter et al. "A roadmap for the computation of persistent homology". In: *EPJ Data Science* 6 (2017), pp. 1–38.

[25] Jan Reininghaus et al. "A stable multi-scale kernel for topological machine learning". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 4741–4748.

[26] Bastian Rieck et al. “Neural Persistence: A Complexity Measure for Deep Neural Networks Using Algebraic Topology”. In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL: https://openreview.net/forum?id=ByxkijC5FQ.

[27] Ann Sizemore, Chad Giusti, and Danielle Bassett. *Classification of weighted networks through mesoscale homological features*. 2016. arXiv: 1512.06457 [math.CO].

[28] Jost Tobias Springenberg et al. “Striving for simplicity: The all convolutional net”. In: *arXiv preprint arXiv:1412.6806* (2014).

[29] Michael Tsang, Dehua Cheng, and Yan Liu. *Detecting Statistical Interactions from Neural Network Weights*. 2018. arXiv: 1705.04977 [stat.ML].

[30] Satoru Watanabe and Hayato Yamana. “Topological measurement of deep neural networks using persistent homology”. In: *Annals of Mathematics and Artificial Intelligence* 90.1 (2022), pp. 75–92.

[31] Nicholas Wells and Chung W See. “Data-driven nonrigid object feature analysis: Subspace application of incidence structure”. In: *Engineering Reports* 2.4 (2020), e12141.

[32] Ji Yang, Lu Sang, and Daniel Cremers. *Dive into Layers: Neural Network Capacity Bounding using Algebraic Geometry*. 2021. arXiv: 2109.01461 [cs.LG].

[33] Matthew D Zeiler and Rob Fergus. “Visualizing and understanding convolutional networks”. In: *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*. Springer. 2014, pp. 818–833.

[34] Chiyuan Zhang et al. “Understanding deep learning (still) requires rethinking generalization”. In: *Communications of the ACM* 64.3 (2021), pp. 107–115.

[35] Umut Şimşekli et al. “Hausdorff dimension, heavy tails, and generalization in neural networks*”. In: *Journal of Statistical Mechanics: Theory and Experiment* 2021.12 (2021), p. 124014. DOI: 10.1088/1742-5468/ac3ae7. URL: https://dx.doi.org/10.1088/1742-5468/ac3ae7.