MASTER'S DEGREE THESIS

Master of Science in Advanced Mathematics and Mathematical Engineering

Title: Improving neural networks using topological data analysis

Author: Rubén Bautista Ballester

Advisor: Julian Pfeifle, Carles Casacuberta, Sergio Escalera

Department: Mathematics

Academic year: 2021 - 2022

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat de Matemàtiques i Estadística

Universitat Politècnica de Catalunya
Facultat de Matemàtiques i Estadística

Master in Advanced Mathematics and Mathematical Engineering

Master's thesis

Improving neural networks using topological data analysis

Rubén Bautista Ballester

Supervised by Julian Pfeifle, Carles Casacuberta, Sergio Escalera

June 2022

Thanks to my mother for everything she has done for me. You are really incredible, I owe you so much. Thanks to Masa for supporting me in some of the toughest decisions I have ever needed to take. Thanks to Julian Pfeifle. Your sharp comments have helped me a lot through the development of this TFM, and I want to keep working with you in the future. Thanks to Karim Lekadir. You have become a very relevant person in my life in the last months. Even though you are always super busy, you always have the time to talk with me when I need it. Your energy is contagious, and I cannot wait to start working on the challenges you have for me! Thanks to Mireia Ribera and Lluís Garrido. You have been teaching me how to become a better professor (and person) during the last year. You are incredible, humble, supportive and smart. Thanks to Aina Ferrà for being part of this family called "Topological Machine Learning @UB" and for being kind of a mentor to me. Thank you all.

Finally, I want to give a special thank you to Carles Casacuberta and Sergio Escalera. Sergio, you have always trusted and supported me no matter what the results were. I'm proud of having you as an advisor, and I owe you so much that a few lines in a TFM are not enough to describe how grateful I am to you. Carles, you really are the mathematician I want to become. I would not have been able to develop this work without your clever comments and our fruitful conversations. You have been there both as an advisor and as a friend for more than a year. Sergio, Carles, I really hope that we can discover the Topological Machine Learning world together during my PhD. Thank you both, really.

Abstract

Generalisation measures are metrics that indicate how well a neural network will perform in presence of unknown data. Differentiable generalisation measures with respect to the parameters of a neural network that use only the training set are candidates to be used as loss regularisation terms to improve neural network training processes. Recently, persistent homology has been used to build robust generalisation measures of this kind by means of persistence diagrams. However, some of these measures involve non-standard distances, and thus the usual stability and differentiability results are not valid. In this thesis, we prove more general stability and differentiability results that fit the conditions required by the previous topological measures. Also, we define a new measure called topological redundancy that we use together with one of the previous topological terms to improve accuracies of networks with respect to usual training without topological regularisation terms.

Keywords

TDA, topological data analysis, topology, metric, distances, differential calculus, persistent homology, persistence diagrams, correlation, machine learning, deep learning, neural networks, regularisation, loss functions, training algorithms.

Contents

1. Introduction

1.1 Motivation

Neural networks are parameterised, computational graph models that are, under some assumptions, able to approximate any reasonable function [31]. This important result is called **universal approximation theorem** and, although it theoretically proves the efficacy of neural networks, it is not possible to build the *ideal* models proposed in it in practice due to several limitations: partial knowledge of the function to approximate, insufficient computational resources, physical limitations, etc. In particular, neural networks are normally used to approximate functions that are known *only* on a limited set of points. Once a graph architecture is fixed and a similarity measure between functions is set, the process of finding the parameter values that yield the best approximation of our target function with the specific neural network architecture is called **training**. The final performance of the network once the parameters are selected is directly related to both the architecture and the training process. However, although the deep learning field has evolved dramatically during the last years, there is no mathematical theory that guides the training and architecture selection processes. This is due to a lack of knowledge of which properties of a network are determinant to produce *better* neural networks given a problem. In fact, the usual approach to build models is based on experimental heuristics, and not on deductive processes.

During the last few years, researchers have been working on solving a simplification of the previous problem called the **generalisation prediction**. Generalisation prediction consists of associating a **generalisation value** given by a **generalisation measure** to each neural network that is directly related to the quality of the approximation of the target function by the neural network. Most state-of-the-art generalisation measures are gathered in the article [23] and in the competition [22], whose results are analysed in the article [21]. In particular, developing precise and robust generalisation measures would allow deep learning practitioners to improve architecture and parameter selection processes to get better approximations. This is crucial in context-critical problems like disease detection or autonomous driving, where a small error of the output can produce dangerous situations.

Differentiable generalisation measures with respect to the parameters of neural networks allows the training process of these networks to be improved. During the training process, a differentiable *measure*, sometimes called **loss function**, between the partially-known target function we want to approximate and the predictions of the neural network for a few known points is minimised. Adding our generalisation measure to this loss function allows us not only to use generic distance measures between the real and predicted points but also to use additional information given by our generalisation measures. This technique is part of a broad set of methods used to improve the training process of a neural network called **regularisation methods**. In particular, we say that we add a **regularisation term** to our loss function when adding a new formula to the original loss function to be minimised.

According to [21], one of the main categories of generalisation measures presented in [22] was related to the study of *intermediate representations* of the input data given by the vertices in the graphs of neural networks. These representations are often analysed by means of geometrical methods. In particular, methods from computational algebraic topology are used in [1] to build generalisation measures capable of obtaining competitive results in predicting the generalisation gap of a neural network, an important measure of generalisation in **classification problems**. A classification problem is usually defined as the problem of approximating a function $f\colon \mathcal{D} \to \mathcal{L} \subset \mathbb{N}$ knowing only a

subset $\mathcal{D}' \subset \mathcal{D}$. We usually do not use the whole set $\mathcal{D}'$ to train our neural network but we split $\mathcal{D}' = \mathcal{D}'_{\text{train}} \cup \mathcal{D}'_{\text{test}}$ to train our neural network with the set $\mathcal{D}'_{\text{train}}$ and to test the resulting approximation with the set $\mathcal{D}'_{\text{test}}$. In particular, the topological methods applied in [1] use only the values of the vertices in the computational graphs when fed with the inputs in the set $\mathcal{D}'_{\text{train}}$ to predict the generalisation gap, a measure computed from both sets $\mathcal{D}'_{\text{train}}$ and $\mathcal{D}'_{\text{test}}$.

The methods from algebraic topology used in [1] belong to a specialty called **topological data analysis**. Topological data analysis (TDA) is a broad field of computational topology that studies the *shape* of *data*, either quantitatively or qualitatively, being persistent homology one of its most prominent subfields. Persistent homology studies how homology groups derived from certain structures, usually finite sets of points, evolve in terms of one or more parameters. Persistent homology is an emerging field that was first introduced in the works [14] and [32] according to [30]. Persistent homology has been broadly used in many areas of science due to its robustness to noise, its descriptive capacity, and the algorithms and data structures that allow to compute and store persistent homology objects easily and quickly in a computer [2, 17, 36].

Robustness of persistent homology methods comes from **stability results**. Stability results state that **persistence diagrams**, the main object of study of persistent homology, are *similar* when the structures yielding the persistence diagrams are *similar*. Stability of persistent homology is itself a big area inside persistent homology with an active community of researchers working on it. For our purposes, the main results of stability were published in [9], where it is proven that persistence diagrams coming from similar sets of points in a metric space with respect to the Gromov-Hausdorff distance are *close* with respect to the bottleneck distance of persistence diagrams, that we define in Section 2.1.4.

On the other hand, the recent article [26] defines a a very general framework for *differentiating* persistence diagrams. Adapting these results to the persistent homology tools used in [1] allows us to use the generalisation measures proposed in the article to build new regularisation terms for neural network loss functions. This is not the first time differentiability of persistence diagrams has been used to improve the loss functions used during a training process. In particular, earlier works on topological machine learning used differentiability without specifying a concrete framework to regularise neural networks by means of new terms in the loss functions. We describe some of them in Section 1.2.

During the thesis, we use persistent homology tools to add regularisation terms based on the results of [1] and [26] to loss functions. We use a new theoretically-based regularisation term called **topological redundancy**, explained in Section 3.2.1, as well as one of the generalisation measures proposed in [1], namely standard deviation of death values in dimension zero persistence diagrams generated from network functional graphs. Finally, we show that topological regularisers can consistently improve the generalisation capacity of neural networks with respect to their accuracy in test. In particular, minimising the standard deviation of deaths in dimension zero persistence diagrams has proved to be the most effective method to regularise neural networks during training, improving the final accuracies in almost all the networks that we have examined and all our experiments. These results are consistent with the plots shown in [1, Figure 4.3], where the lower the standard deviation is, the better the generalisation capacity of the neural network is.

1.2 State of the art

As mentioned in the previous section, several approaches have been undertaken on adding topological regularisers to loss functions.

In [10], persistent homology is applied mainly to binary classification problems ($\mathcal{L} = \{l_1, l_2\}$) in which neural networks represent functions $\mathcal{N}\colon \mathcal{D} \to \mathbb{R}$ where an input x is classified as label l_1 if $\mathcal{N}(x) < 0$ and as l_2 otherwise, where the set $\mathcal{N}^{-1}(0)$ is called the decision boundary of the neural network $\mathcal{N}$ and the connected components of $\mathcal{D} \setminus \mathcal{N}^{-1}(0)$ are called decision regions. In particular, they develop a regularisation term based on persistent homology that penalises weak decision regions, that is, decision regions $\mathcal{R}$ where $\max_{x\in\mathcal{R}} |x|$ is *near* zero.

In [20], a topological regulariser is used in an image segmentation context. Image segmentation is the process of assigning labels to all the pixels of an input image, e.g., detecting the pixels of a heart in an image. Usually, these labels are related with geometrical properties, like having an specific number of connected components or handles. Some of these geometrical properties can be tracked with (persistent) homology. In this paper, they add to the usual loss functions for image segmentation problems a regulariser that rewards conserving persistent homology features of the known segmentations in the neural network segmentation predictions. Similar work is done in [20], where a prior description of the topological structure is known and the topological regulariser tries to fit the neural network predictions to have the desired topological structure. Finally, in [19] authors apply discrete Morse theory and persistent homology on image segmentation neural networks to identify important *topological structures* during training and to use these structures to improve the results of the neural network.

In [15] we find a framework of persistent homology differentiability and its implementation in the Python package PyTorch [29]. The framework is tested successfully by minimising/maximising certain persistent homology descriptors in three different problems: topological noise reduction and regularisation, optimisation of generative models, and generation of adversarial attacks. This framework is generalised in the subsequent articles [6, 26], that are the first ones to provide an explicit framework for differential calculus in persistence diagrams.

In [18], the authors use persistent homology to build a differentiable regularisation term that allows to control topological and geometric properties of an autoencoder's latent space. An autoencoder is a type of neural network $\mathcal{N}\colon \mathcal{D} \to \mathcal{S}$ whose objective is to learn *better* representations of inputs $\mathcal{I} \subseteq \mathcal{D}$ with respect to some property, like the dimension of $\mathcal{D}$ and $\mathcal{S}$. The output space is called the autoencoder's latent space. In particular, they find that pairwise distances between representation of points $x, y \in \mathcal{I}$ in the autoencoder's latent space $\mathcal{N}(x), \mathcal{N}(y) \in \mathcal{S}$ are close to a fixed distance η, showing that this property is beneficial for these kind of neural networks.

In [7], the authors apply differentiability of persistence diagrams to generative models. Generative models are neural networks that, given an input, that may be even noise, produce an output with some desired properties (like being an image of a cat) satisfied in a set of examples $\mathcal{D}$. In particular, they penalise that (*big*) sets of generated outputs do not share the persistent homology descriptors yielded by the set of original examples.

1.3 Contributions

The main objective of this thesis is to develop new regularisation terms based on the previous work [1] that can help to choose better parameters of a neural network during training. To do that, we use persistence diagrams coming from Vietoris-Rips filtrations, that are nested families of Vietoris-Rips simplicial complexes (2.14). Vietoris-Rips filtrations are common for practical purposes due to the efficient algorithms developed to compute them [2, 17, 36]. However, stability results have been mainly proven for Vietoris-Rips-based persistence diagrams coming from metric spaces [9, 13]. Also,

the differential calculus framework for persistence diagrams developed in [26] consider only Vietoris-Rips persistence diagrams in Euclidean metric spaces. This is a limitation because many practical applications use similarity functions to compute distances between objects of interest. In our case, we follow the framework developed in [1], that applies persistent homology in vectors coming from vertices in the neural network where the way to measure a distance between vectors is given by the function $d(x, y) = 1 - |\text{Corr}(x, y)|$. In this work, we

1. prove stability results for Vietoris-Rips filtrations using similarity functions that are not necessarily metrics under some mild assumptions that the function $1 - |\text{Corr}(x, y)|$ satisfies;

2. develop the differentiability framework of persistence diagrams published in [26] for general Vietoris-Rips filtrations using a similarity function instead of the Euclidean distance;

3. formalise, unify and adapt the theory of neural networks and deep learning using the theory of computational graphs to use the persistent homology framework presented in [1];

4. build successful regularisation terms based on both theoretical and empirical considerations about neural network properties captured by persistence diagrams using the framework of [1].

The first two contributions are developed in Subection 2.2 whereas the last two (contributions 3 and 4), are developed in Subsection 3.2 and Section 4, respectively.

1.4 Outline

The thesis is divided into the following sections: Introduction 1, Persistence Stability and Differentiability 2, Deep Learning 3, Experimental Results 4, and Conclusions 5.

The Introduction contains a brief motivation of the work 1.1, a review of the state of the art 1.2, and the contributions done in this thesis 1.3.

The Persistence Stability and Differentiability section contains an introduction of persistent homology 2.1 and its classical results, including the differentiability results of [26] in 2.1.5. After that, our own results on stability and differentiability for similarity functions are developed in Subsection 2.2.

The Deep Learning section contains a formalisation of the main terms of deep learning and neural networks 3.2 based on the theory of computational graphs, presented in 3.1. Lastly, a new topological regularisation term is presented in 3.2.1.

The Experimental Results section contains details of the experiments performed and their results. The first three subsections include a definition of the regularisation terms we use in our experiments 4.1, a basic explanation of how dropout works 4.2, and a way to plot meaningful graphics of dimension zero persistence diagrams with density functions 4.3. The last two subsections are dedicated to the description of the experiments performed and their results, in Subsections 4.4 and 4.5 respectively.

Finally, the Conclusions section contains a discussion about the results 5 and possible future work related to this thesis 5.1.

2. Persistence stability and differentiability

2.1 Persistent homology

2.1.1 Fundamentals of persistence modules

In this chapter we define persistent modules and persistence diagrams from finite sets of points with an associated distance. Most definitions and results are extracted from [9].

Definition 2.1. ([9, Definition 1.1]) Let $\mathbb{K}$ be a field and $\mathbb{R}$ the set of real numbers. A persistence module over $\mathbb{K}$ is an $\mathbb{R}$-indexed family of $\mathbb{K}$-vector spaces

$$\mathbb{V} = \{V_t : t \in \mathbb{R}\}$$

together with a doubly-indexed family of linear maps

$$\left\{v_s^t \colon V_s \to V_t \mid s, t \in \mathbb{R},\ s \leq t\right\}$$

which satisfy the composition law

$$v_s^t \circ v_r^s = v_r^t$$

whenever $r \leq s \leq t$, and where v_t^t is the identity map on V_t for all t.

We say that a persistence module $\mathbb{V}$ is pointwise finite-dimensional if V_t has finite dimension for every t.

Equivalently, a persistence module can be regarded as a functor from the partially ordered set $(\mathbb{R}, \leq)$ to the category $\mathbf{Vect}_{\mathbb{K}}$ of vector spaces over $\mathbb{K}$. This category-theoretical point of view is useful in many instances. For example, it leads to a short definition of a morphism of persistence modules, and consequently of an isomorphism of persistence modules, as follows.

Definition 2.2. ([9, Definition 1.3]) A morphism $\eta \colon \mathbb{V} \to \mathbb{W}$ between persistence modules $\mathbb{V}$ and $\mathbb{W}$ is a natural transformation of persistence modules regarded as functors. It is an isomorphism if η is an isomorphism of functors.

The category of persistence modules **Pers** has persistence modules as objects and morphisms between them as arrows.

In the next definition, an interval is any connected subset of $\mathbb{R}$ with the usual topology.

Definition 2.3. ([9, Definition 1.4]) Let $J \subseteq \mathbb{R}$ be an interval and $\mathbb{K}$ a field. We denote by $\mathbb{I}_J$ the persistence module over $\mathbb{K}$ with

$$I_t = \begin{cases} \mathbb{K} & \text{if } t \in J, \\ 0 & \text{otherwise,} \end{cases} \qquad i_s^t = \begin{cases} \mathrm{Id}_{\mathbb{K}} & \text{if } s, t \in J, \\ 0 & \text{otherwise.} \end{cases}$$

Definition 2.4. ([9, Definition 1.5]) The direct sum $\mathbb{W} = \mathbb{U} \oplus \mathbb{V}$ of two persistence modules $\mathbb{U}$, $\mathbb{V}$ is defined as

$$W_t = U_t \oplus V_t, \qquad w_s^t = u_s^t \oplus v_s^t \qquad \text{for all } s \leq t.$$

Theorem 2.5 (Crawley-Boevey [12])**.** *Any pointwise finite-dimensional persistence module* $\mathbb{V}$ *over a field* $\mathbb{K}$ *has a unique decomposition as a direct sum of interval modules, that is, there exists a unique multi-set* $\mathcal{J}$ *of intervals of* $\mathbb{R}$ *such that*

$$\mathbb{V} \cong \bigoplus_{J \in \mathcal{J}} \mathbb{I}_J. \tag{1}$$

For persistence modules decomposed as in Theorem 2.5, the multi-set $\mathcal{J}$ is called the barcode of $\mathbb{V}$, where the interval modules are depicted as a collection of parallel horizontal bars. Barcodes can alternatively be represented as persistence diagrams, which are defined next.

Definition 2.6. ([9, Definition 1.6]) The persistence diagram of a pointwise finite-dimensional persistence module $\mathbb{V} \cong \bigoplus_{J \in \mathcal{J}} \mathbb{I}_J$ is the multiset

$$\mathrm{Dgm}(\mathbb{V}) = \{(\inf J, \sup J) \,:\, J \in \mathcal{J}\} \setminus \Delta^{\infty}, \tag{2}$$

where Δ^{∞} is a multiset containing countably many copies of the diagonal

$$\Delta = \{(x, x) : x \in \mathbb{R} \cup \{-\infty, +\infty\}\}.$$

We denote the set of all persistence diagrams as **Bar**. For points $(b, d) \in \mathrm{Dgm}(\mathbb{V})$, we denote the coordinates b and d by birth and death of the point (a, b), respectively. The multisets of all these births and deaths for all the points in a persistence diagram are called simply births and deaths of the persistence diagram, respectively.

Although it is not needed to develop the theory of persistence modules as in [9], it is usual in the topological data analysis community to consider persistence diagrams to also contain Δ^{∞}. In the context of our work, adding the diagonal simplifies some proofs.

Definition 2.7. The extended persistence diagram of a persistence module $\mathbb{V}$ is the multiset

$$\mathrm{Dgm}_{\Delta}(\mathbb{V}) = \mathrm{Dgm}(\mathbb{V}) \cup \Delta^{\infty}. \tag{3}$$

We denote the set of all extended persistence diagrams by $\mathbf{Bar}_{\Delta}$.

2.1.2 Simplicial complexes and simplicial homology

The following definitions are central to our work.

Definition 2.8. ([13, Section III.1]) An abstract simplicial complex is a finite collection of sets K such that if $\alpha \in K$ and $\beta \subseteq \alpha$ then $\beta \in K$. The sets of an abstract simplicial complex are called simplices. A subset of an abstract simplicial complex that is also an abstract simplicial complex is called a subcomplex.

We will be using the terms *simplicial complex* and *abstract simplicial complex* indistinctly to refer to the same notion 2.8.

Definition 2.9. ([26, Definition 2.7]) Let K be an abstract simplicial complex. A filter function on K is a function $f : K \to \mathbb{R}$ that is monotonous with respect to inclusions of faces in K, i.e.,

$$\forall \sigma, \sigma' \in K, \qquad \sigma \subseteq \sigma' \implies f(\sigma) \leq f(\sigma'). \tag{4}$$

We denote the set of filter functions on a given simplicial complex K as $\mathbb{R}^K$.

Definition 2.10. Let K be a simplicial complex and $f \in \mathbb{R}^K$ a filter function on K. The sublevel set at level t is the set $K^t = \{\sigma \in K : f(\sigma) \leq t\}$. By monotonicity of filter functions, K^t is a subcomplex of the simplicial complex K and also of every other sublevel set K^s with $t \leq s$.

Inclusions between sublevel sets will be denoted as

$$i_t^s \colon K^t \to K^s, \quad t \leq s.$$

These inclusions satisfy that

$$i_t^s \circ i_r^t = i_r^s$$

whenever $r \leq t \leq s$, and $i_t^t = \mathrm{Id}_{K^t}$. For a simplicial complex K and a filter function $f \in \mathbb{R}^K$, the category of all sublevel sets and inclusion maps is called the sublevel set filtration of the pair (K, f), and it is denoted K_{sub} or K_{sub}^f.

From a sublevel set filtration we obtain a persistence module by applying any functor from simplicial complexes to vector spaces. Let $H_{p,\mathbb{K}}$ be the functor that assigns to each simplicial complex its p-dimensional simplicial homology group with coefficient field $\mathbb{K}$, and to each inclusion map the induced homomorphism on p-dimensional homology, as in [13]. We will work with the field $\mathbb{Z}_2$ of two elements unless otherwise specified, and we will omit the symbol $\mathbb{K}$ in our notation from now on.

Since we are considering finite simplicial complexes, the vector spaces spanned by their simplices are finitely generated, and thus their homology groups are finitely generated too. This implies the following fact.

Proposition 2.11. *For every simplicial complex K, the persistence module $H_p(K_{\mathrm{sub}}^f)$ is pointwise finite-dimensional for all $f \in \mathbb{R}^K$ and $p \geq 0$.*

It follows that the structure of our persistence modules $H_p(K_{\mathrm{sub}}^f)$ is uniquely determined by Theorem 2.5. From now on we will focus only on this kind of persistence modules. We will denote the persistence diagram $\mathrm{Dgm}(H_p(K_{\mathrm{sub}}^f))$ as $\mathrm{Dgm}_p(K_{\mathrm{sub}}^f)$ to make visible the homology dimension in which we work. Also, when referring to extended persistence diagrams we will use $\mathrm{Dgm}_{p,\Delta}(K_{\mathrm{sub}}^f)$ to denote $\mathrm{Dgm}_\Delta(H_p(K_{\mathrm{sub}}^f))$.

2.1.3 Vietoris-Rips, point clouds and distance matrices

There are many ways to construct simplicial complexes and filter functions. In real-world applications, practitioners focus especially on building simplicial complexes and filter functions from point clouds equipped with a similarity function, where usually each point of the point cloud represents a measure from an entity in a system which they want to know about. For this purpose, there are several canonical constructions such as Čech complexes, Delaunay complexes or alpha complexes, to name a few [13]. However, Vietoris-Rips complexes and their filter functions are the most used ones in practice due to the availability of efficient algorithms [2, 17, 36] to compute persistence diagrams coming from their persistence modules.

Definition 2.12. A finite ordered set (P, o) is a set P of cardinality n equipped with a bijective map $o \colon \{1, \dots, n\} \to P$ called ordering. We denote $p_i = o(i)$ for all $i \in \{1, \dots, n\}$.

Definition 2.13. A point cloud is a finite ordered set P equipped with a function $d \colon P \times P \to \mathbb{R}$ such that $d(a, b) = d(b, a)$ for all a, b.

In most cases, but not necessarily, the function d will satisfy the **distance** axioms. We use the same symbol d to denote the function $\{1,\dots,n\}\times\{1,\dots,n\} : (i,j)\mapsto d(p_i,p_j)\in\mathbb{R}$ induced by d. We denote by $\mathcal{P}(\{1,\dots,n\})$ the set of all subsets of $\{1,\dots,n\}$.

Definition 2.14. Let (P,d) be a point cloud with $|P|=n$ and $K=\mathcal{P}(\{1,\dots,n\})\setminus\{\emptyset\}$. Let $r\in\mathbb{R}$. The Vietoris-Rips complex of P of radius r is the simplicial complex $K^r(P)$ given by

$$K^r(P)=\{\sigma\subseteq\{1,\dots,n\}\mid \operatorname{diam}\sigma\leq r,\ \sigma\neq\emptyset\}, \tag{5}$$

where $\operatorname{diam}\sigma=\max_{i,j\in\sigma}d(i,j)$.

The simplicial complex $K^r(P)$ can be seen as a sublevel set of the filter function $f\colon K\to\mathbb{R}$ that assigns to each simplex of K its diameter, i.e.,

$$f(\sigma)=\operatorname{diam}\sigma=\max_{i,j\in\sigma}d(i,j). \tag{6}$$

This function f is monotone under simplex inclusions, since if $\sigma\subseteq\sigma'$ then $\max_{i,j\in\sigma}d(i,j)\leq\max_{i,j\in\sigma'}d(i,j)$.

Definition 2.15. The Vietoris-Rips filtration of the point cloud (P,d) with $|P|=n$ is the sublevel set filtration given by the simplicial complex $K=\mathcal{P}(\{1,\dots,n\})\setminus\{\emptyset\}$ and the filter function f defined in Definition 2.14.

We will denote this sublevel set filtration as $VR_d(P)$ and its persistence diagrams of a given dimension p as $\mathrm{Dgm}_p(VR_d(P))$ and $\mathrm{Dgm}_{p,\Delta}(VR_d(P))$ for the usual and extended ones, respectively.

2.1.4 Classical stability results

Some of the most important results about persistence modules coming from point clouds and persistence diagrams are the called stability theorems. These theorems guarantee that, under *small perturbations* of the input data, the resulting persistence diagrams are *near*. This is somewhat similar to continuity in ordinary functions, and it is needed to guarantee, for example, that the methods are robust when the input is altered by little quantities of noise, among others. There are many results of this kind of stability, among which we will be using the ones in Chazal et al. articles [8, 9].

Stability is usually proven for metric spaces —in our case, point clouds (P,d) where d is a metric on P. These results can be extended and proved in several ways, as in [8]. On one hand, some methods use persistence diagrams and partial matchings to prove stability. On the other hand, other methods rely on having extended persistence diagrams and bijections between them. For our purposes, both ways yield equivalent results, although the ones involving only persistence diagrams are usually more sophisticated, as the ones in [9]. In our case, as we will be using the differentiability results of the article [26] where authors use extended persistence diagrams, we will also take this approach.

First we need to define when two persistence diagrams are *close*. To do this, we use the classical metric between persistence diagrams, the bottleneck distance, that is a particularisation of a more general concept called Wasserstein distance [13].

Definition 2.16. ([26, Definition 2.10]) Given two barcodes $D,D'\in\mathbf{Bar}_\Delta$, viewed as multi-sets, a matching is a bijection $\gamma\colon D\to D'$. The cost of γ is

$$c(\gamma)=\sup_{x\in D}\|x-\gamma(x)\|_\infty\in\mathbb{R}\cup\{+\infty\}, \tag{7}$$

and the set of all matchings between D and D' is denoted by $\Gamma(D,D')$.

Definition 2.17. ([26, Definition 2.11]) The bottleneck distance between two barcodes $D, D' \in \mathbf{Bar}_\Delta$ is defined as

$$d_\infty(D, D') = \inf_{\gamma \in \Gamma(D,D')} c(\gamma). \tag{8}$$

We want to bound this bottleneck distance between two persistence diagrams D and D' from the *distance* between the point clouds we used to generate them. However, the point clouds may be completely different in terms of their ambient spaces (metrics, definitions, etc). A classical way to compare such point clouds is to use the Gromov-Hausdorff distance, as in [8]. Although the original definition of the Gromov-Hausdorff distance [28]

$$d_{GH}(X, Y) = \inf_{Z,f,g} d_H^Z(f(X), g(Y)), \quad d_H^Z = \max\left(\sup_{a\in A}\inf_{b\in B} d(a, b), \sup_{b\in B}\inf_{a\in A} d(a, b)\right),$$

where $f : X \to Z$, $g : Y \to Z$ are isometric embeddings into the metric space (Z, d), is not the one we are going to work with here, it can be proved that both notions are equivalent in metric spaces [4].

Definition 2.18. ([8, Definition 4.1]) A correspondence $C : X \rightrightarrows Y$ is a surjective multivalued map from X to Y, that is, a subset $C \subseteq X \times Y$ where for all $x_0 \in X$ there is some $(x_0, y) \in C$ and for all $y_0 \in Y$ there is some $(x, y_0) \in C$.

Definition 2.19. Let (X, d_X), (Y, d_Y) be metric spaces. The Gromov-Hausdorff distance between them is

$$d_{GH}(X, Y) = \frac{1}{2}\inf\{\mathrm{dis}(C) : \ C \text{ is a correspondence } X \rightrightarrows Y\}, \tag{9}$$

where

$$\mathrm{dis}(C) = \sup\left\{\left|d_X(x, x') - d_Y(y, y')\right| : \ (x, y), (x', y') \in C\right\} \tag{10}$$

is called the distortion of the correspondence C.

Note that if the function d defining our point cloud is a metric, then (P, d) is a finite metric space (a metric space whose set has finite cardinality). Therefore, the following classical stability result holds on our point clouds.

Theorem 2.20. ([8, Theorem 5.2]) *If (X, d_X) and (Y, d_Y) are finite metric spaces, then*

$$d_\infty(\mathrm{Dgm_p}(\mathrm{VR_{d_X}}(X)), \mathrm{Dgm_p}(\mathrm{VR_{d_Y}}(Y))) \leq 2\, \mathrm{d_{GH}}(X, Y). \tag{11}$$

Note that we have been working so far with metric spaces. Recall that in real data applications we often use similarity measures that do not satisfy the distance axioms, but are useful to gain insights about how close two arbitrary elements in a domain of interest are. We will be analysing this phenomenon in Subsection 2.2.1.

2.1.5 Differentiability of functions on persistence diagrams

Persistence diagrams usually give useful information about the data we are handling. Being able to impose a certain persistence diagram structure on our data would imply that we also would be able to require some topological properties on these data. If persistence diagrams, in some way, were differentiable with respect to our point clouds, then we would be able to impose these structures by minimising/maximising functions with the points of the persistence diagram as input. This is precisely what we define in this section: a framework for differential calculus on persistence diagrams

developed by Leygonie et al. [26] that will be the first step to minimise/maximise some functions that indicate how well a neural network generalise.

Throughout this section, $\mathcal{M}$ denotes a smooth finite-dimensional manifold without boundary, which may or may not be compact. We will base the differentiability of a function $\mathcal{M} \to \mathbf{Bar}_\Delta$ on the factorisation of this function through an space of *ordered* persistence diagrams.

Definition 2.21. For each choice of $m, n \in \mathbb{Z}_{\geq 0}$, the space of ordered barcodes with m finite points and n infinite ones is $\mathbb{R}^{2m} \times \mathbb{R}^n$ equipped with the Euclidean norm and the usual smooth structure.

Each *finite point* in the space of ordered barcodes represents a point in an extended persistence diagram whose origin is a bounded interval in $\mathbb{R}$. On the other hand, each *infinite point* in $\mathbb{R}$ represents a point whose origin is an interval only bounded by below. By definition of extended persistence diagrams, there are four types of points:

1. Totally bounded points: (a, b) with $-\infty < a \leq b < +\infty$.
2. Lower bounded points: $(a, +\infty)$ with $a < +\infty$.
3. Upper bounded points: $(-\infty, b)$ with $-\infty < b$.
4. Unbounded points: $(-\infty, +\infty)$.

We restrict ourselves to persistence diagrams that only have lower bounded points ((1) and (2)), which are the ones yielded by the majority of classical sublevel set filtrations, including Vietoris-Rips filtrations of point clouds as they generate finite type persistence modules 2.47.Intuitively, our vectors in $\mathbb{R}^{2m} \times \mathbb{R}^n$ represent the *important points* (most of them off-diagonal) of a persistence diagram and give them an order to work with.

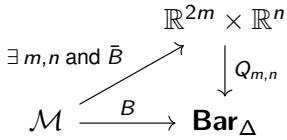
Our first objective is to differentiate representatives of persistence diagrams in the space of ordered barcodes. To do so, we need a projection from our space of ordered barcodes that yields persistence diagrams in $\mathbf{Bar}_\Delta$.

Definition 2.22. ([26, Definition 3.1]) Let $\bar{D} = (b_1, d_1, \dots, a_m, d_m, v_1, \dots, v_n) \in \mathbb{R}^{2m} \times \mathbb{R}^n$ be an ordered barcode. The quotient map $Q_{m,n} \colon \mathbb{R}^{2m} \times \mathbb{R}^n \to \mathbf{Bar}_\Delta$ is a map that projects $\bar{D}$ to the space of extended persistence diagrams, forgetting the order of the points, that is

$$Q_{m,n}\left(\bar{D}\right) = \{(b_i, d_i)\}_{i=1}^{m} \cup \{(v_j, +\infty)\}_{j=1}^{n} \cup \Delta^\infty. \tag{12}$$

With these two new tools in hand, we can define when a function $\mathcal{M} \to \mathbf{Bar}_\Delta$ is differentiable.

Definition 2.23. ([26, Definition 3.3]) Let $B \colon \mathcal{M} \to \mathbf{Bar}_\Delta$ be a persistence diagram valued map. Let $x \in \mathcal{M}$ and $r \in \mathbb{Z}_{\geq 0} \cup \{+\infty\}$. We say that B is r-differentiable at x if there exists an open neighbourhood U of x, integers $m, n \in \mathbb{Z}_{\geq 0}$ and a map $\bar{B} \colon U \to \mathbb{R}^{2m} \times \mathbb{R}^n$ of class $\mathcal{C}^r$ such that $B = Q_{m,n} \circ \bar{B}$ on U. For an integer $d \in \mathbb{Z}_{\geq 0}$, a function $\mathcal{B} \colon \mathcal{M} \to \mathbf{Bar}_\Delta^{d+1}$ is r-differentiable at $x \in \mathcal{M}$ if each of its $d+1$ components is r-differentiable. $\bar{B}$ is called a local lift of B. We have



If a map $B\colon \mathcal{M} \to \mathbf{Bar}_\Delta$ is r-differentiable with $r \geq 1$ we can also define its differential.

Definition 2.24. ([26, Definition 3.6]) Let $B\colon \mathcal{M} \to \mathbf{Bar}_\Delta$ be 1-differentiable at some $x \in \mathcal{M}$. Let $\bar{B}\colon U \to \mathbb{R}^{2m} \times \mathbb{R}^n$ be a $\mathcal{C}^1$ lift of B defined on an open neighbourhood U of x. The differential $d_{x,\bar{B}}B$ of B at x with respect to $\bar{B}$ is the differential of $\bar{B}$ at x

$$T_x\mathcal{M} \xrightarrow[d_x\bar{B}]{} \mathbb{R}^{2m} \times \mathbb{R}^n. \tag{13}$$

We are not only interested in the differentiability of persistence diagrams but also in the differentiability of functions coming from it, like the classical total persistence function defined as

$$\mathcal{T}(D) = \sum_{(b,d)\in D\setminus\Delta^\infty \text{ s.t. } b,d\notin\{-\infty,+\infty\}} d - b.$$

Let us now define the differentiability of functions the other way around, from our space of persistence diagrams to any smooth finite-dimensional manifold without boundary $\mathcal{N}$.

Definition 2.25. ([26, Definition 3.10]) Let $V\colon \mathbf{Bar}_\Delta \to \mathcal{N}$ be a map on persistence diagrams. Let $D \in \mathbf{Bar}_\Delta$ and $r \in \mathbb{Z}_{\geq 0} \cup \{+\infty\}$. V is said to be r-differentiable at D if for all integers $m, n \in \mathbb{Z}_{\geq 0}$ and all vectors $\bar{D} \in \mathbb{R}^{2m} \times \mathbb{R}^n$ such that $Q_{m,n}\left(\bar{D}\right) = D$, the map $V \circ Q_{m,n}\colon \mathbb{R}^{2m} \times \mathbb{R}^n \to \mathcal{N}$ is $\mathcal{C}^r$ on an open neighbourhood of $\bar{D}$.

Again, we define the differential of a r-differentiable function $V\colon \mathbf{Bar}_\Delta \to \mathcal{N}$ for $r \geq 1$.

Definition 2.26. ([26, Definition 3.13]) Let $V\colon \mathbf{Bar}_\Delta \to \mathcal{N}$ be 1-differentiable at $D \in \mathbf{Bar}_\Delta$ and $\bar{D} \in \mathbb{R}^{2m} \times \mathbb{R}^n$ be a pre-image of D via $Q_{m,n}$. The differential of V at D with respect to $\bar{D}$ is the map

$$d_{D,\bar{D}}V\colon \mathbb{R}^{2m} \times \mathbb{R}^n \xrightarrow[d_{\bar{D}}(V\circ Q_{m,n})]{} T_{V(D)}\mathcal{N}.$$

We can combine the previous definitions to give results about differentiability for functions $\mathcal{M} \to \mathcal{N}$ that factor through $\mathbf{Bar}_\Delta$, i.e., functions $\mathcal{M} \to \mathbf{Bar}_\Delta \to \mathcal{N}$.

Proposition 2.27. ([26, Proposition 3.14]) *Let $B\colon \mathcal{M} \to \mathbf{Bar}_\Delta$ be r-differentiable at $x \in \mathcal{M}$ and $V\colon \mathbf{Bar}_\Delta \to \mathcal{N}$ be r-differentiable at $B(x)$. Then:*

1. *$V \circ B\colon \mathcal{M} \to \mathcal{N}$ is $\mathcal{C}^r$ at x as a map between smooth manifolds.*
2. *If $r \geq 1$, then for any local $\mathcal{C}^1$ lift $\bar{B}\colon U \to \mathbb{R}^{2m} \times \mathbb{R}^n$ of B around x we have:*

$$d_x\left(V \circ B\right) = d_{B(x),\bar{B}(x)}V \circ d_{x,\bar{B}}B. \tag{14}$$

The relation (14) shows that even though the differentials of B and V may depend on the choice of the lift $\bar{B}$, their composition does not, and they work together as a usual differential between smooth manifolds.

2.1.6 Differentiability through Vietoris-Rips filtrations

Let us now analyse the differentiability of persistence diagrams coming from Vietoris-Rips subleve set filtrations. The proofs of this section require many technical details that we will not use through the thesis. For this reason, we only state the fundamental results that we need and we refer the interested reader to the source of this differential framework [26].

We are going to state the results for Vietoris-Rips sublevel set filtrations coming from point clouds (P, d) whose ambient space is the usual Euclidean space $\left(\mathbb{R}^d, m(x,y) = \|x-y\|_2\right)$ with its usual smooth structure. For our purposes, we need also to fix the cardinality of the point clouds to a number n, i.e., $|P| = n$. Therefore, we can see our point clouds as points P such that $P = (p_1, \dots, p_n) \in \mathbb{R}^{nd}$ also with its usual smooth structure.

In our case, we will define functions B of the form

$$B_p : \mathcal{M} = \mathbb{R}^{nd} \xrightarrow{F} \mathbb{R}^K \xrightarrow{\mathrm{Dgm}_{p,\Delta}} \mathbf{Bar}_\Delta, \tag{15}$$

where p denotes the homology dimension, $K = \mathcal{P}\left(\{1, \dots, n\}\right) \setminus \{\emptyset\}$ and $F(P)(\sigma) = \max_{i,j\in\sigma}\|p_i - p_j\|_2$ as in Section 2.1.3. Now, we state that there exists a generic set of points $\bar{\mathcal{P}} \subseteq \mathbb{R}^{nd}$ where the maps B_p are ∞-differentiable for all p.

Definition 2.28. ([26, Definition 5.6]) A point cloud $P = (p_1, \dots, p_n) \in \mathbb{R}^{nd}$ is in general position if the following two conditions hold:

1. For all $i \neq j \in \{1, \dots, n\}$ we have $p_i \neq p_j$.
2. For all $\{i, j\} \neq \{k, l\}$ where $i, j, k, l \in \{1, \dots, n\}$, $\|p_i - p_j\|_2 \neq \|p_k - p_l\|_2$.

We denote the set of point clouds in general position by $\bar{\mathcal{P}} \subseteq \mathbb{R}^{nd}$.

Proposition 2.29. ([26, Proposition 5.7]) *$\bar{\mathcal{P}}$ is generic in $\mathbb{R}^{nd}$.*

Proposition 2.30. ([26, Proposition 5.8]) *The parametrization $F : \mathbb{R}^{nd} \to \mathbb{R}^K$ is C^∞ over $\bar{\mathcal{P}}$.*

Finally, we state the key result for differentiability of Vietoris-Rips filtrations coming from point clouds.

Corollary 2.31. ([26, Corollary 5.9]) *The barcode valued map $B_p : P \in \mathbb{R}^{nd} \mapsto \mathrm{Dgm}_{p,\Delta}(F(P)) \in \mathbf{Bar}_\Delta$ is ∞-differentiable in $\bar{\mathcal{P}}$.*

2.2 Beyond metric spaces

2.2.1 Stability with symmetric functions

In Section 2.1.4 we stated a stability result for point clouds equipped with a distance function. However, in many scenarios, the distances between objects we care about do not satisfy all the necessary properties to be true distances. In these cases, there is no general classical result that can be applied everywhere. For some specific kinds of filter functions, there are known results that generalise the stability theorems we have seen so far, as [8, Lemma 4.9], that generalises stability results for Dowker filtrations $\mathrm{Dow}(\Lambda)$, a very general way of building sublevel set filtrations based on the Dowker

complexes $\text{Dow}(\Lambda, a)$ defined by $\sigma \in \text{Dow}(\Lambda, a) \iff \exists w \in W$ such that $\Lambda(l, w) \leq a$ for all $l \in \sigma$, where L, W are two non-empty, arbitrary sets and $\Lambda \colon L \times W \to \mathbb{R}$ is any function.

In this section we prove stability results based on mild assumptions about a symmetric function of a point cloud (P, d'). In particular, we are interested in stability results for point clouds with $P \subset \mathbb{R}^m$, with m a fixed constant, and $d' \colon (x, y) \in \mathbb{R}^m \times \mathbb{R}^m \mapsto 1 - |\text{Corr}(x, y)| \in \mathbb{R}$, that was proven to give useful information about the generalisation capacity of a neural network using TDA techniques in [1]. Unfortunately, it was proved in [34] that, although $d(x, y) = \sqrt{1 - \text{Corr}^2(x, y)}$ is a distance, $d'(x, y) = 1 - |\text{Corr}(x, y)|$ is not. However, note that d and d' are closely related. In particular, we can show the following result.

Lemma 2.32. *Let $f \colon P \times P \to \mathbb{R}$ be any function. Define $d \colon (x, y) \in P \times P \to \sqrt{1 - f^2(x, y)} \in \mathbb{R}$, $d' \colon (x, y) \in P \times P \to 1 - |f(x, y)| \in \mathbb{R}$, and $\gamma(x) \colon x \in \mathbb{R} \to 1 - \sqrt{1 - x^2} \in \mathbb{R}$. Then $d' = \gamma \circ d$.*

Proof.

$$(\gamma \circ d)(x, y) = 1 - \sqrt{1 - \left(\sqrt{1 - f^2(x, y)}\right)^2} = 1 - \sqrt{1 - (1 - f^2(x, y))} = 1 - |f(x, y)| = d'(x, y).$$

□

This means that if we substitute $f(x, y)$ by our correlation function we obtain a transformation from one true distance function to our symmetric function. We know that in fact, for our distance function d, the stability result 2.20 holds. However, our symmetric function d' is only a transformation of this distance. We will see that under reasonable regularity assumptions on γ, we can prove our desired stability result for functions $\gamma \circ d$ where d is any distance function.

The main property that functions γ must satisfy to state our new stability results is the Hölder continuity condition.

Definition 2.33. Let $C, \alpha \in \mathbb{R}_{>0}$. A function $f \colon \text{Dom}(f) \subseteq \mathbb{R} \to \mathbb{R}$ is (C, α)-Hölder continuous if

$$|f(x) - f(y)| \leq C\,|x - y|^{\alpha} \tag{16}$$

for all $x, y \in \text{Dom}(f)$.

From now on, let X and Y be two finite sets in a metric space $(\mathcal{M}, d)$ and let d' be a symmetric function $d' \colon \mathcal{M} \times \mathcal{M} \to \mathbb{R}$. Let $\gamma \colon \text{Im}(d) \subseteq \mathbb{R}_{\geq 0} \longrightarrow \mathbb{R}$ a strictly increasing, (C, α)-Hölder continuous function for some $C, \alpha > 0$, such that $d' = \gamma \circ d$. These are our conditions of **regularity** for the function γ.

During the whole section we introduce new notations to ease the reading of the proofs. Denote $D_p(X) = \text{Dgm}_p(VR_d(X))$, $D_{p,\Delta}(X) = \text{Dgm}_{p,\Delta}(VR_d(X))$, $D'_p(X) = \text{Dgm}_p(VR_{d'}(X))$ and $D'_{p,\Delta}(X) = \text{Dgm}_{p,\Delta}(VR_{d'}(X))$. We write $D(X)$, $D'(X)$, $D_\Delta(X)$ and $D'_\Delta(X)$ to refer to $D_p(X)$, $D'_p(X)$, $D_{p,\Delta}(X)$ and $D'_{p,\Delta}(X)$ respectively when the dimension of persistence diagrams do not influence the discussion, i.e., the results hold for any dimension.

The first result that we prove is that persistence modules yielded by the Vietoris-Rips sublevel set filtrations for point clouds with common set P and distances d and d' are the same but *shifted*, i.e., $K^t_d = \{\sigma \in K : f_d(\sigma) \leq t\} = K^{\gamma(t)}_{d'} = \{\sigma \in K : f_{d'}(\sigma) \leq \gamma(t)\}$ with f_d, $f_{d'}$ the filter functions for the Vietoris-Rips filtrations with distances d and d', respectively and $K = \mathcal{P}(\{1, \dots, |X|\}) \setminus \emptyset$. To prove this result, we need an easy lemma.

Lemma 2.34. $\gamma(a) \leq \gamma(b) \iff a \leq b$.

Proof.

- $\implies$]: Suppose not. Then, there exists $a > b$ such that $\gamma(a) \leq \gamma(b)$. However, γ is strictly increasing, so we get $\gamma(a) > \gamma(b)$, that is a contradiction.
- $\impliedby$]: If $a = b$ then $\gamma(a) = \gamma(b)$ because γ is a function. If $a < b$ then, as γ is strictly increasing, we have $\gamma(a) < \gamma(b)$.

□

Proposition 2.35. *Let $f_d, f_{d'}$ be the Vietoris-Rips filter functions given by the functions $d, d' : X \times X \to \mathbb{R}$ respectively as in 2.14. Then, $f_{d'}(\sigma) = \gamma(f_d(\sigma))$ and $K_d^t = K_{d'}^{\gamma(t)}$ where $K = \mathcal{P}(\{1, \dots, |X|\}) \setminus \emptyset$.*

Proof. In particular, we have by definition that $f_d(\sigma) = \max_{i,j\in\sigma} d(i,j)$ and $f_{d'}(\sigma) = \max_{i,j\in\sigma} d'(i,j)$ and that $f_d, f_{d'} \in \mathbb{R}^K$. The first part of the proposition follows because γ is an increasing function so we have

$$f_{d'}(\sigma) = \max_{i,j\in\sigma} d'(i,j) = \max_{i,j\in\sigma} (\gamma \circ d)(i,j) \underbrace{=}_{\gamma \text{ increasing}} \gamma(\max_{i,j\in\sigma} d(i,j)) = \gamma(f_d(\sigma)). \tag{17}$$

Let's see now that $K_d^t = K_{d'}^{\gamma(t)}$.

- $\subseteq$]: If $\sigma \in K_d^t$ then $f_d(\sigma) \leq t$. By Equation (17) we have $f_{d'}(\sigma) = \gamma(\underbrace{f_d(\sigma)}_{\leq t}) \underbrace{\leq}_{\gamma \text{ increasing}} \gamma(t)$. Therefore, $\sigma \in K_{d'}^{\gamma(t)}$.
- $\supseteq$]: If $\sigma \in K_{d'}^{\gamma(t)}$ then $f_{d'}(\sigma) \leq \gamma(t)$. On one hand, we have by Equation (17) that $f_{d'}(\sigma) = \gamma(f_d(\sigma))$. On the other hand, as $f_{d'}(\sigma) \leq \gamma(t)$ we get

$$f_{d'}(\sigma) = \gamma(f_d(\sigma)) \leq \gamma(t) \underbrace{\implies}_{\text{Lemma 2.34}} f_d(\sigma) \leq t.$$

□

The previous result is a form of *reindexing* an original filtration by a function γ, and it is used by Ripser [2] to compute persistence diagrams from point clouds efficiently. Reindexings are defined using filtrations, an abstraction of the sublevel set filtration concept.

Definition 2.36. ([2, Section 2]) Given a finite simplicial complex K, a filtration of K is a collection of subcomplexes $F = (K^i)_{i\in I}$ of K where I is a totally ordered indexing set such that $i \leq j$ implies $K^i \subseteq K^j$. We can see a filtration as a functor $F^\bullet : I \to \mathbf{Simp}$ from I as a poset category to **Simp**, the category of simplicial complexes where the arrows $i < j$ are sent to the inclusion maps $K_i \subseteq K_j$.

Note that our sublevel set filtrations are general filtrations indexed by $(\mathbb{R}, \leq)$. We use this more general concept to build special, generally discrete, filtrations that allow us to give a characterisation of persistence diagrams of sublevel set filtrations.

Definition 2.37. ([2, Section 2]) A filtration is essential if $i \neq j$ implies $K_i \neq K_j$. A simplexwise filtration of K is a filtration such that, for all $i \in I$ with $K_i \neq \emptyset$ there is some simplex $\sigma_i \in K$ and some index $j < i \in I$ such that $K_i \setminus K_j = \{\sigma_i\}$. In an essential simplexwise filtration, the index j is the predecessor of i in I.

Definition 2.38. ([2, Section 2]) A reindexing of a filtration $F^\bullet \colon R \to \mathbf{Simp}$ indexed over some totally ordered set R is another filtration $K^\bullet \colon I \to \mathbf{Simp}$ such that $F_t = K_{r(t)}$ for some monotonic map $r \colon R \to I$ called reindexing map. If there is a complex K_i that does not occur in the filtration $F^\bullet$, we say that $K^\bullet$ refines $F^\bullet$.

The following proposition gives a characterisation of persistence diagrams of a filtration in terms of essential simplexwise refinements of sublevel set filtrations.

Proposition 2.39. ([2, Proposition 2.1]) *Let $f : K \to \mathbb{R}$ be a filter function on a simplicial complex K and let $K^\bullet : I \to \mathbf{Simp}$ be an essential simplexwise refinement of the sublevel set filtration $F^\bullet = f^{-1}(-\infty, \bullet]$ with $K_i = \{\sigma_k \mid k \in I, k \leq i\}$. Denote by $H_p(F^\bullet)$ the persistence module generated by applying the homology functor $H_{p,\mathbb{K}}$ from simplicial complexes to vector spaces for any fixed field $\mathbb{K}$ as in Section 2.1.2. The barcode, as in 2.5, of $K^\bullet$, $\mathcal{J}(H_p(F^\bullet))$, determines the barcode of $F^\bullet$, $\mathcal{J}(H_p(K^\bullet))$, i.e.,*

$$\mathcal{J}(H_p(F^\bullet)) = \left\{ r^{-1}[i,j) \neq \emptyset \mid [i,j) \in \mathcal{J}(H_p(K^\bullet)) \right\}, \tag{18}$$

with $r^{-1}[i,j) = [f(\sigma_i), f(\sigma_j))$ and $r^{-1}[i,+\infty) = [f(\sigma_i), +\infty)$.

The existence of essential simplexwise refinements of the sublevel set filtrations yielded by Vietoris-Rips filtrations of point clouds is given in [2].

With Proposition 2.39 we are able to prove a fundamental relation of the persistence diagrams yielded by Vietoris-Rips filtrations with functions d and d'.

Corollary 2.40. *$D'_p(X) = \{(\gamma(a), \gamma(b)) : (a,b) \in D_p(x)\}$ with $\gamma(+\infty) = +\infty$ for any homology dimension p.*

Proof. Take an essential simplexwise refinement $K^\bullet : I \to \mathbf{Simp}$ of the sublevel set filtration $VR_d(X)$ with reindexing map r as in Proposition 2.39. This is also an essential simplexwise refinement of the sublevel set filtration $VR_{d'}(X)$ by taking $r \circ \gamma^{-1}$ as reindexing map because $K^t_d = K^{\gamma(t)}_{d'}$ by Proposition 2.35. The reindexing map is well-defined because γ^{-1} is well-defined in $Dom\, d'$ and also because γ^{-1} is strictly increasing because it is the inverse of a strictly increasing function, so $r \circ \gamma^{-1}$ is monotone from $(\mathbb{R}, \leq)$ to $(I, \leq)$ by composition of monotone functions. Recall from Definition 2.14 that $\mathrm{diam}_d(\sigma) = \max_{i,j \in \sigma} d(i,j)$. Therefore, we have

$$\begin{aligned}
\mathcal{J}(VR_{d'}(X)) &= \left\{ \left(r \circ \gamma^{-1}\right)^{-1}[i,j) \neq \emptyset \mid [i,j) \in \mathcal{J}(H_p(K^\bullet)) \right\} \\
&= \left\{ \left(\gamma \circ r^{-1}\right)[i,j) \neq \emptyset \mid [i,j) \in \mathcal{J}(H_p(K^\bullet)) \right\} \\
&= \left\{ \gamma\left(r^{-1}([i,j))\right) \neq \emptyset \mid [i,j) \in \mathcal{J}(H_p(K^\bullet)) \right\} \\
&= \left\{ [\gamma(\mathrm{diam}_d(\sigma_i)), \gamma(\mathrm{diam}_d(\sigma_j))) : [i,j) \in \mathcal{J}(H_p(K^\bullet)) \wedge i,j < +\infty \right\} \\
&\cup \left\{ [\gamma(\mathrm{diam}_d(\sigma_i)), +\infty) : [i,+\infty) \in \mathcal{J}(H_p(K^\bullet)) \wedge i < +\infty \right\},
\end{aligned}$$

with

$$\begin{aligned}
\mathcal{J}(VR_d(X)) &= \left\{ r^{-1}[i,j) \neq \emptyset \mid [i,j) \in \mathcal{J}(H_p(K^\bullet)) \right\} \\
&= \left\{ [\mathrm{diam}_d(\sigma_i), \mathrm{diam}_d(\sigma_j)) : [i,j) \in \mathcal{J}(H_p(K^\bullet)) \wedge i,j < +\infty \right\} \\
&\cup \left\{ [\mathrm{diam}_d(\sigma_i), +\infty) : [i,+\infty) \in \mathcal{J}(H_p(K^\bullet)) \wedge i < +\infty \right\}.
\end{aligned}$$

As γ is bijective, we have

$$[a, b) \neq \emptyset \iff [\gamma(a), \gamma(b)) \neq \emptyset.$$

Then, we have a one-to-one correspondence between points in $\mathcal{J}(VR_d(X))$ and in $\mathcal{J}(VR_{d'}(X))$. In particular, from the previous three equations we obtain

$$\mathcal{J}(VR_{d'}(X)) = \bigcup_{[a,b)\in\mathcal{J}(VR_d(X))} [\gamma(a), \gamma(b)).$$

Note that $(\inf[a, b), \sup[a, b)) = (a, b)$ for all intervals $[a, b) \subseteq \mathbb{R}$. Therefore, by definition of persistence diagram 2.6, we get the desired result, $D'_p(X) = \{(\gamma(a), \gamma(b)) : (a, b) \in D_p(x)\}$. □

By definition of extended persistence diagram 2.7 and the previous Corollary 2.40 the following corollary follows immediately.

Corollary 2.41. $D'_\Delta(X) = \{(\gamma(a), \gamma(b)) : (a, b) \in D(x)\} \cup \Delta^\infty$.

Now we are going to use bijections from $D_\Delta(X)$ to $D_\Delta(Y)$ to bound the bottleneck distance between $D'_\Delta(X)$ and $D'_\Delta(Y)$. We are interested only in bijections with low cost in the bottleneck distance between $D_\Delta(X)$ and $D_\Delta(Y)$ as in 2.16. For this reason, we will work only with matchings $\iota : D(X) \to D(Y)$ such that $(a, b) \mapsto (\frac{a+b}{2}, \frac{a+b}{2})$ if (a, b) is sent by ι to a point in the diagonal. This is because they get always lower costs than matchings not satisfying this property.

Lemma 2.42. *For any matching $\iota : D_\Delta(X) \to D_\Delta(Y)$ not satisfying $(a, b) \mapsto (\frac{a+b}{2}, \frac{a+b}{2})$ if $\iota(a, b) \in \Delta^\infty$, we can find another matching $\iota' : D_\Delta(X) \to D_\Delta(Y)$ satisfying the previous property such that*

$$\sup_{(a,b)\in D_\Delta(X)} \|(a, b) - \iota'(a, b)\|_\infty \leq \sup_{(a,b)\in D_\Delta(X)} \|(a, b) - \iota(a, b)\|_\infty.$$

Proof. It is easy to see that

$$\arg\min_{d\in\mathbb{R}\cup\{+\infty\}} \|(a, b) - (d, d)\|_\infty = \frac{a+b}{2}, \tag{19}$$

for $a, b < +\infty$. If $a < \infty$ and $b = +\infty$, then

$$\|(a, b) - (d, d)\|_\infty = \max(|a - d|, |+\infty - d|).$$

If $d < +\infty$ then,

$$\|(a, b) - (d, d)\|_\infty = \max(|a - d|, +\infty) = +\infty,$$

whereas if $d = +\infty$

$$\|(a, b) - (d, d)\|_\infty = \max(+\infty, 0) = +\infty,$$

so the norm is infinite for each choice of d and in particular

$$\frac{a+b}{2} = +\infty \in \operatorname{args\,min}_{d\in\mathbb{R}\cup\{+\infty\}} \|(a, b) - (d, d)\|_\infty. \tag{20}$$

Now let us suppose that there exists (a, b) such that $\iota(a, b) = (d, d)$ with $d \neq \frac{a+b}{2}$. Define ι' such that $\iota'(\alpha, \beta) = \iota(\alpha, \beta)$ if $(\alpha, \beta) \neq (a, b)$, $\iota'(a, b) = (\frac{a+b}{2}, \frac{a+b}{2})$. Also, as there are infinitely many points in the diagonal we can set $\iota'^{-1}(\iota(a, b)) = \iota'^{-1}(d, d) = (d, d) \in \Delta^\infty$. Therefore, ι' is a matching and we get

$$\sup_{(a,b)\in D_\Delta(X)} \|(a, b) - \iota'(a, b)\|_\infty \leq \sup_{(a,b)\in D_\Delta(X)} \|(a, b) - \iota(a, b)\|_\infty,$$

because $\|(\alpha, \beta) - \iota'(\alpha, \beta)\|_\infty = \|(\alpha, \beta) - \iota(\alpha, \beta)\|_\infty$ if $(\alpha, \beta) \neq (a, b)$ and, by Equations (19) and (20), $\|(a, b) - \iota'(a, b)\|_\infty \leq \|(a, b) - \iota(a, b)\|_\infty$. □

Note that if $\iota : D \to D'$ is a matching then $\iota^{-1}\colon D' \to D$ is a matching too so we can apply Corollary 2.42 to have *equivalent* matchings $\iota' : D \to D'$ such that all the off-diagonal and diagonal points (a, b) sent to diagonal points of the other persistence diagram are especifically sent to points $\left(\frac{a+b}{2}, \frac{a+b}{2}\right)$. The following lemma is the main tool we need to define our stability result for d'.

Lemma 2.43. *Let $D_\Delta(X)$ and $D_\Delta(Y)$ be two extended persistence diagrams. For any matching $\iota : D_\Delta(X) \to D_\Delta(Y)$ such that $\iota(a, b) \in \Delta^\infty$ implies $\iota(a, b) = \left(\frac{a+b}{2}, \frac{a+b}{2}\right)$ and such that $\iota^{-1}(a, b) \in \Delta^\infty$ implies $\iota^{-1}(a, b) = \left(\frac{a+b}{2}, \frac{a+b}{2}\right)$, there exists a matching $\bar{\iota}\colon D'_\Delta(X) \to D'_\Delta(Y)$ with*

$$\sup_{(a,b)\in D_\Delta(X)} \|(a, b) - \bar{\iota}((a, b))\|_\infty \leq \beta \cdot \sup^{\alpha}_{(a,b)\in D'_\Delta(X)} \|(a, b) - \iota((a, b))\|_\infty,$$

where $\beta = \max\left(C, C \cdot 2^{\alpha-1}\right)$.

Proof. We prove this lemma by means of partial matchings. A partial matching is a map m between multisets $\mathrm{Dom}(m) \subseteq D'(X) \overset{m}{\mapsto} D'(Y)$. Our matching $\bar{\iota}$ is defined such that $\bar{\iota}(a, b) = m(a, b)$ if m is defined in (a, b) and $\iota(a, b) = \iota^{-1}(a, b) = \left(\frac{a+b}{2}, \frac{a+b}{2}\right)$ otherwise. Note that all points in $D'_\Delta(X)$ and $D'_\Delta(Y)$ are matched with points of the contrary diagram, including diagonal points, due to we have inifinitely many points in the diagonals to map unmatched points by m. Adjusting the matches with the diagonal points to have one-to-one correspondences we get a bijective matching between $D'_\Delta(X)$ and $D'_\Delta(Y)$. Define then

$$m(a, b) = \left(\gamma(\iota(\gamma^{-1}(a))), \gamma(\iota(\gamma^{-1}(b)))\right),$$

for points $(a, b) \in D'(X)$ such that $\left(\gamma(\iota(\gamma^{-1}(a))), \gamma(\iota(\gamma^{-1}(b)))\right) \in D'(Y)$. What we are doing here is to transform matches $(a, b) \in D(X) \overset{\iota}{\mapsto} (c, d) \in D(Y)$ to matches $(\gamma(a), \gamma(b)) \in D'(X) \overset{\iota'}{\mapsto} (\gamma(c), \gamma(d)) \in D'(Y)$ by means of m. We claim that $\bar{\iota}$ satisfies the desired property.

Define $(\iota(a), \iota(b)) = \iota(a, b)$. Suppose $(a, b) \in \mathrm{Dom}(m)$ and thus $\bar{\iota}(a, b) = m(a, b)$. Then, we have four possibilities for $\bar{\iota}$.

1. $(a, b) \mapsto (\bar{\iota}(a), \bar{\iota}(b))$ with $-\infty < a, b, \bar{\iota}(a), \bar{\iota}(b) < +\infty$. By Corollary 2.40, we have $-\infty < \gamma^{-1}(a), \gamma^{-1}(b), \iota(\gamma^{-1}(a)), \iota(\gamma^{-1}(b)) < +\infty$. Therefore, we have:

$$\begin{aligned}
\|(a, b) - \bar{\iota}(a, b)\|_\infty &= \left\|(a, b) - \left(\gamma(\iota(\gamma^{-1}(a))), \gamma(\iota(\gamma^{-1}(b)))\right)\right\|_\infty \\
&= \max\left(\left|a - \gamma(\iota(\gamma^{-1}(a)))\right|, \left|b - \gamma(\iota(\gamma^{-1}(b)))\right|\right) \\
&= \max\left(\left|\gamma(\gamma^{-1}(a)) - \gamma(\iota(\gamma^{-1}(a)))\right|, \left|\gamma(\gamma^{-1}(b)) - \gamma(\iota(\gamma^{-1}(b)))\right|\right) \\
&\underbrace{\leq}_{(C,\alpha)\text{-Hölder}} \max\left(C \cdot \left|\gamma^{-1}(a) - \iota(\gamma^{-1}(a))\right|^\alpha, C \cdot \left|\gamma^{-1}(b) - \iota(\gamma^{-1}(b))\right|^\alpha\right) \\
&\underbrace{=}_{C,\alpha>0} C \cdot max^\alpha\left(\left|\gamma^{-1}(a) - \iota(\gamma^{-1}(a))\right|, \left|\gamma^{-1}(b) - \iota(\gamma^{-1}(b))\right|\right) \\
&= C \cdot \left\|\left(\gamma^{-1}(a), \gamma^{-1}(b)\right) - \iota\left(\gamma^{-1}(a), \gamma^{-1}(b)\right)\right\|^\alpha_\infty.
\end{aligned}$$

2. $(a, b) \mapsto (\bar{\iota}(a), \bar{\iota}(b) = +\infty)$ with $-\infty < a, b, \bar{\iota}(a) < +\infty$. By Corollary 2.40, we have $\iota(\gamma^{-1}(b)) = +\infty$ and $-\infty < \gamma^{-1}(a), \gamma^{-1}(b), \iota(\gamma^{-1}(a)) < +\infty$. Therefore, we have

$$\begin{aligned}
\left\|\left(\gamma^{-1}(a), \gamma^{-1}(b)\right) - \left(\iota\left(\gamma^{-1}(a)\right), +\infty\right)\right\|_\infty &= \max\left(|\gamma^{-1}(a) - \iota\left(\gamma^{-1}(a)\right)|, |\gamma^{-1}(b) - \infty|\right) \\
&= \max\left(|\gamma^{-1}(a) - \iota\left(\gamma^{-1}(a)\right)|, +\infty\right) = +\infty,
\end{aligned}$$

   that implies

$$\|(a, b) - (\bar{\iota}(a), +\infty)\|_\infty \leq C \cdot \left\|\left(\gamma^{-1}(a), \gamma^{-1}(b)\right) - \left(\iota\left(\gamma^{-1}(a)\right), +\infty\right)\right\|^\alpha_\infty = +\infty.$$

3. $(a, b = +\infty) \mapsto (\bar{\iota}(a), \bar{\iota}(b))$ with $-\infty < a, \bar{\iota}(a), \bar{\iota}(b) < +\infty$. By Corollary 2.40, we have $\gamma^{-1}(b) \underbrace{=}_{b=+\infty} +\infty$ and $-\infty < \gamma^{-1}(a), \iota(\gamma^{-1}(a)), \iota(\gamma^{-1}(b)) < +\infty$. Therefore, we have

$$\begin{aligned}\left\|(\gamma^{-1}(a), \gamma^{-1}(b)) - \iota(\gamma^{-1}(a), \gamma^{-1}(b))\right\|_\infty &= \left\|(\gamma^{-1}(a), +\infty) - (\iota(\gamma^{-1}(a)), \iota(\gamma^{-1}(b)))\right\|_\infty \\ &= \max\left(|\gamma^{-1}(a) - \iota(\gamma^{-1}(a))|, |\infty - \iota(\gamma^{-1}(b))|\right) \\ &= \max\left(|\gamma^{-1}(a) - \iota(\gamma^{-1}(a))|, +\infty\right) = +\infty,\end{aligned}$$

that implies

$$\|(a, b) - \bar{\iota}(a, b)\|_\infty \leq C \cdot \left\|(\gamma^{-1}(a), \gamma^{-1}(b)) - \iota(\gamma^{-1}(a), \gamma^{-1}(b))\right\|_\infty^\alpha = +\infty.$$

4. $(a, b = +\infty) \mapsto (\bar{\iota}(a), \bar{\iota}(b) = +\infty)$ with $-\infty < a, \bar{\iota}(a) < +\infty$. By Corollary 2.40, we have $\gamma^{-1}(b) \underbrace{=}_{b=+\infty} +\infty \underbrace{=}_{\bar{\iota}(b)=+\infty} \iota(\gamma^{-1}(b)) = +\infty$ and $-\infty < \gamma^{-1}(a), \iota(\gamma^{-1}(a)) < +\infty$. Then,

$$\begin{aligned}\|(a, b) - (\bar{\iota}(a), \bar{\iota}(b))\|_\infty &= \left\|(a, +\infty) - \left(\gamma(\iota(\gamma^{-1}(a))), +\infty\right)\right\|_\infty \\ &= \max\left(|a - \gamma(\iota(\gamma^{-1}(a)))|, 0\right) \\ &= |a - \gamma(\iota(\gamma^{-1}(a)))| \\ &= \left|\gamma\left(\gamma^{-1}(a)\right) - \gamma(\iota(\gamma^{-1}(a)))\right| \leq C \cdot \left|\gamma^{-1}(a) - \iota(\gamma^{-1}(a))\right|^\alpha \\ &= C \cdot \left\|(\gamma^{-1}(a), \gamma^{-1}(b)) - \left(\iota\left(\gamma^{-1}(a)\right), \iota\left(\gamma^{-1}(b)\right)\right)\right\|_\infty^\alpha.\end{aligned}$$

Suppose now $(a, b) \notin \mathrm{Dom}(m)$. Therefore we have the following possibilities for points $(a, b) \in D'_\Delta(X)$.

1. $(a, b) \in D'(X) \setminus \mathrm{Dom}(m)$. This means that (a, b) is a point $\left(\gamma^{-1}(a), \gamma^{-1}(b)\right) \in D(X)$ paired to a point in the diagonal of $D(Y)$. We have two cases:

   - a, b such that $a < +\infty$ and $b = +\infty$. By Corollary 2.40 we have $\gamma^{-1}(a) < +\infty$ and $\gamma^{-1}(b) = +\infty$. Also, by hypothesis

   $$\left(\gamma^{-1}(a), \gamma^{-1}(b)\right) \mapsto \left(\frac{\gamma^{-1}(a) + \gamma^{-1}(b)}{2}, \frac{\gamma^{-1}(a) + \gamma^{-1}(b)}{2}\right) = (+\infty, +\infty).$$

   We get

   $$\begin{aligned}\left\|\left(\gamma^{-1}(a), \gamma^{-1}(b)\right) - \iota\left(\gamma^{-1}(a), \gamma^{-1}(b)\right)\right\|_\infty &= \max\left(\left|\gamma^{-1}(a) - \infty\right|, \left|\gamma^{-1}(b) - \infty\right|\right) \\ &= \max\left(+\infty, 0\right) = +\infty.\end{aligned} \tag{21}$$

   Therefore,

   $$\|(a, b) - \bar{\iota}(a, b)\|_\infty \leq C \cdot \left\|\left(\gamma^{-1}(a), \gamma^{-1}(b)\right) - \iota\left(\gamma^{-1}(a), \gamma^{-1}(b)\right)\right\|_\infty^\alpha = +\infty.$$

   - $a, b < +\infty$. By Corollary 2.40 we have $\gamma^{-1}(a), \gamma^{-1}(b) < +\infty$ and by hypothesis

   $$\left(\gamma^{-1}(a), \gamma^{-1}(b)\right) \mapsto \left(\frac{\gamma^{-1}(a) + \gamma^{-1}(b)}{2}, \frac{\gamma^{-1}(a) + \gamma^{-1}(b)}{2}\right)$$

We get

$$\left\|\left(\gamma^{-1}(a),\gamma^{-1}(b)\right)-\left(\frac{\gamma^{-1}(a)+\gamma^{-1}(b)}{2},\frac{\gamma^{-1}(a)+\gamma^{-1}(b)}{2}\right)\right\|_\infty=\frac{\left|\gamma^{-1}(b)-\gamma^{-1}(a)\right|}{2}. \quad (22)$$

On the other hand, with $\bar{\iota}$, we pair $(a,b)\mapsto\left(\frac{a+b}{2},\frac{a+b}{2}\right)$, yielding

$$\begin{aligned}\|(a,b)-\bar{\iota}(a,b)\|_\infty &= \left\|(a,b)-\left(\frac{a+b}{2},\frac{a+b}{2}\right)\right\|_\infty\\ &= \frac{|b-a|}{2}=\frac{\left|\gamma(\gamma^{-1}(b))-\gamma(\gamma^{-1}(a))\right|}{2}\\ &\leq C\cdot\frac{\left|\gamma^{-1}(b)-\gamma^{-1}(a)\right|^\alpha}{2}=C\cdot 2^{\alpha-1}\cdot\left(\frac{\left|\gamma^{-1}(b)-\gamma^{-1}(a)\right|}{2}\right)^\alpha\\ &\underbrace{=}_{\text{Eq. 22}} C\cdot 2^{\alpha-1}\cdot\left\|\left(\gamma^{-1}(a),\gamma^{-1}(b)\right)-\left(\frac{\gamma^{-1}(a)+\gamma^{-1}(b)}{2},\frac{\gamma^{-1}(a)+\gamma^{-1}(b)}{2}\right)\right\|_\infty^\alpha\\ &= C\cdot 2^{\alpha-1}\cdot\left\|\left(\gamma^{-1}(a),\gamma^{-1}(b)\right)-\iota\left(\gamma^{-1}(a),\gamma^{-1}(b)\right)\right\|_\infty^\alpha.\end{aligned}$$

2. $(a,b)\notin D'(X)$. Then, we have two possibilities.

   (a) $(a,b)\in\Delta^\infty$ diagonal point of $D'_\Delta(X)$ paired with a point $(c,d)\in D'(Y)$. This means that $\left(\gamma^{-1}(c),\gamma^{-1}(d)\right)\in D(Y)$ is paired with the diagonal in $D_\Delta(X)$. By symmetry with the previous case 1 we have

   $$\begin{aligned}&\|(a,b)-\bar{\iota}(a,b)\|_\infty\leq\max\left(C,C\cdot 2^{\alpha-1}\right)\cdot\left\|\iota^{-1}\left(\gamma^{-1}(c),\gamma^{-1}(d)\right)-\left(\gamma^{-1}(c),\gamma^{-1}(d)\right)\right\|_\infty^\alpha\\ &=\max\left(C,C\cdot 2^{\alpha-1}\right)\cdot\left\|\left(\iota^{-1}\left(\gamma^{-1}(c)\right),\iota^{-1}\left(\gamma^{-1}(d)\right)\right)-\iota\left(\iota^{-1}\left(\gamma^{-1}(c)\right),\iota^{-1}\left(\gamma^{-1}(d)\right)\right)\right\|_\infty^\alpha.\end{aligned}$$

   (b) $(a,b)\in\Delta^\infty$ diagonal point paired with itself in the other diagram ($\frac{a+b}{2}=\frac{2a}{2}=a$ for $a=b<+\infty$ or $a=b=+\infty$ and $\frac{a+b}{2}=+\infty$), yielding

   $$\|(a,b)-\bar{\iota}(a,b)\|_\infty=0. \quad (23)$$

Take $\beta=\max\left(C,C\cdot 2^{\alpha-1}\right)$. Note that $\sup\left(A^\alpha\right)\leq(\sup A)^\alpha$ because x^α is a continuous and strictly increasing function for $\alpha>0$. Therefore, by the previous inequalities, we get

$$\sup_{(a,b)}\|(a,b)-\bar{\iota}((a,b))\|_\infty\leq\beta\cdot\sup^\alpha_{(a,b)}\|(a,b)-\iota((a,b))\|_\infty,$$

□

From now on take $\beta=\max\left(C,C\cdot 2^{\alpha-1}\right)$. Note that β only depends on α, that is fixed when γ is given. Before stating the main result of this section, we need to see that the infimum of Definition 2.17 can be replaced by a minimum.

Corollary 2.44. *Let $D,D'\in$ **Bar** two persistence diagrams with $|D|,|D'|<+\infty$. Let $D_\Delta=D\cup\Delta^\infty$, $D'_\Delta=D'\cup\Delta^\infty$ in $\mathbf{Bar}_\Delta$ be their extended versions. Define*

$$\begin{aligned}\Gamma'\left(D_\Delta,D'_\Delta\right)=\Big\{\iota\in\Gamma\left(D_\Delta,D'_\Delta\right):\iota(a,b)\in\Delta^\infty\implies\iota(a,b)=\left(\frac{a+b}{2},\frac{a+b}{2}\right)\wedge\\ \iota^{-1}(a,b)\in\Delta^\infty\implies\iota^{-1}(a,b)=\left(\frac{a+b}{2},\frac{a+b}{2}\right)\Big\}\subseteq\Gamma\left(D_\Delta,D'_\Delta\right).\end{aligned}$$

The bottleneck distance for D_Δ and D'_Δ satisfies

$$d_\infty\left(D_\Delta, D'_\Delta\right) = \min_{\iota\in\Gamma'\left(D_\Delta, D'_\Delta\right)} c(\iota).$$

Proof. Set $\Gamma = \Gamma\left(D_\Delta, D'_\Delta\right)$ and $\Gamma' = \Gamma'\left(D_\Delta, D'_\Delta\right)$. We have $\Gamma' \subseteq \Gamma$. By Corollary 2.42 we get that $\inf_{\iota\in\Gamma'} c(\iota) = \inf_{\iota\in\Gamma} c(\iota)$. However, notice that D and D' have finite cardinality and also there is a finite number of diagonal points that can be matched to points in D, D' for matchings $\iota \in \Gamma'$. Notice also that all diagonal points of D_Δ and D'_Δ not matched with points in D' and D respectively are matched to themselves in the other diagram and thus have cost zero. Therefore, there is a finite number of possible costs $c(\iota)$ for $\iota \in \Gamma'$, i.e. $|\{c(\iota) : \iota \in \Gamma'\}| < +\infty$ and therefore the infimum is reached in Γ' so

$$d_\infty\left(D_\Delta, D'_\Delta\right) = \inf_{\iota\in\Gamma\left(D_\Delta, D'_\Delta\right)} c(\iota) = \inf_{\iota\in\Gamma'\left(D_\Delta, D'_\Delta\right)} c(\iota) = \min_{\iota\in\Gamma'\left(D_\Delta, D'_\Delta\right)} c(\iota).$$

□

Vietoris-Rips persistence diagrams have finite cardinality because Vietoris-Rips persistence modules are of finite type.

Definition 2.45. ([5, Section 5.1]) We say that a persistence module $\mathbb{V}$ over a field $\mathbb{K}$ is of finite type if it consists of a finite number of finite-dimensional vector spaces.

Lemma 2.46. *Let (P, d) be a point cloud and $p \in \mathbb{N}$. Vietoris-Rips persistence modules $VR_d(P)$ are of finite type.*

Proof. We need to show that we have a finite number of different vector spaces of finite dimension. The second property, having vector spaces with finite dimension, holds for all persistence modules coming from sublevel set filtrations and the homology functor by Proposition 2.11, so it holds also in our particular case. It remains to be shown that we only generate a finite number of different vector spaces. However, this comes directly from our definition of Vietoris-Rips sublevel set filtration, as we apply the homology functor on subsets of $K = \mathcal{P}\left(\{1, \dots, n\}\right) \setminus \{\emptyset\}$ and there are a finite number of subsets of K, that yield a finite number of possible different vector spaces. □

Finite type persistence modules have finite decompositions as in Theorem 2.5.

Proposition 2.47. ([5, Proposition 3]) *A finite type persistence module $\mathbb{V}$ over a field $\mathbb{K}$ has a unique decomposition*

$$\mathbb{V} \cong \bigoplus_{J\in\mathcal{J}} \mathbb{I}_J, \tag{24}$$

as in Theorem 2.5 such that $|\mathcal{J}| < +\infty$ and such that every interval $J \in \mathcal{J}$ is bounded from below.

A direct consequence is that persistence diagrams of finite type have finite cardinality.

Corollary 2.48. *For a finite type persistence module $\mathbb{V}$ we have $|\mathrm{Dgm}(\mathbb{V})| < +\infty$.*

Proof. By definition, we have $\mathrm{Dgm}(\mathbb{V}) = \{(\inf J, \sup J) \,:\, J \in \mathcal{J}\} \setminus \Delta^\infty$ and by Proposition 2.47, $|\mathcal{J}| < +\infty$ so $|\mathrm{Dgm}(\mathbb{V})| < +\infty$. □

The following is the main result of this section.

Corollary 2.49. $d_\infty\left(D'_\Delta(X), D'_\Delta(Y)\right) \leq \beta \cdot d_\infty^\alpha\left(D_\Delta(X), D_\Delta(Y)\right)$.

Proof. By Lemma 2.46, Corollary 2.48 and Corollary 2.44 we have that the bottleneck distances for the persistence diagrams $D'_\Delta(X)$ and $D'_\Delta(Y)$, and $D_\Delta(X)$ and $D_\Delta(Y)$, satisfy

$$d_\infty\left(D'_\Delta(X), D'_\Delta(Y)\right) = \min_{\iota \in \Gamma'\left(D'_\Delta(X), D'_\Delta(Y)\right)} c(\iota),$$
$$d_\infty\left(D_\Delta(X), D_\Delta(Y)\right) = \min_{\iota \in \Gamma'\left(D_\Delta(X), D_\Delta(Y)\right)} c(\iota).$$

Note that any matching $\iota \in \Gamma'\left(D_\Delta(X), D_\Delta(Y)\right)$ satisfies the assumptions of Lemma 2.43. Also, the resulting matching $\bar{\iota}$ given by Lemma 2.43 satisfies $\bar{\iota} \in \Gamma'\left(D'_\Delta(X), D'_\Delta(Y)\right)$ and

$$c\left(\bar{\iota}\right) = \sup_{(a,b)\in D'_\Delta(X)} \|(a,b) - \bar{\iota}((a,b))\|_\infty \leq \beta \cdot \sup^\alpha_{(a,b)\in D_\Delta(X)} \|(a,b) - \iota((a,b))\|_\infty = \beta \cdot c^\alpha(\iota). \quad (25)$$

Take $\iota' \in \Gamma'\left(D_\Delta(X), D_\Delta(Y)\right)$ attaining the minimum of the bottleneck distance, i.e., $d_\infty\left(D_\Delta(X), D_\Delta(Y)\right) = c(\iota')$. Therefore, by Equation (25) we have

$$c\left(\bar{\iota'}\right) \leq \beta \cdot c^\alpha\left(\iota'\right) = \beta \cdot d_\infty^\alpha\left(D_\Delta(X), D_\Delta(Y)\right). \quad (26)$$

Finally, as $\bar{\iota'} \in \Gamma'\left(D'_\Delta(X), D'_\Delta(Y)\right)$, we have

$$d_\infty\left(D'_\Delta(X), D'_\Delta(Y)\right) = \min_{\iota \in \Gamma'\left(D'_\Delta(X), D'_\Delta(Y)\right)} c(\iota) \leq c\left(\bar{\iota'}\right) \underbrace{\leq}_{(26)} \beta \cdot d_\infty^\alpha\left(D_\Delta(X), D_\Delta(Y)\right).$$

□

Now, we are able to state our stability theorem for functions $d' = \gamma \circ d$.

Theorem 2.50. *Let X, Y be two finite sets in a metric space $(\mathcal{M}, d)$. Let $d' \colon \mathcal{M} \times \mathcal{M} \to \mathbb{R}$ be a symmetric function and $\gamma \colon \mathrm{Im}(d) \subseteq \mathbb{R}_{\geq 0} \longrightarrow \mathbb{R}$ be a strictly increasing, (C, α)-Hölder continuous function for some $C, \alpha > 0$, such that $d' = \gamma \circ d$. Then it is satisfied*

$$d_\infty\left(D'_\Delta(X), D'_\Delta(Y)\right) \leq 2^\alpha \beta \cdot \left(d^d_{GH}(X, Y)\right)^\alpha,$$

where $\beta = \max\left(C, C \cdot 2^{\alpha-1}\right)$ and $D'_\Delta(X), D'_\Delta(Y) \in \mathbf{Bar}_\Delta$ are the extended persistence diagrams $\mathrm{Dgm}_{p,\Delta}(\mathrm{VR}_{d'}(X))$, $\mathrm{Dgm}_{p,\Delta}(\mathrm{VR}_{d'}(Y))$ for any dimension $p \in \mathbb{Z}_{\geq 0}$, respectively.

Proof. By the stability theorem 2.20 we have

$$d_\infty\left(D_\Delta(X), D_\Delta(Y)\right) \leq 2 d^d_{GH}(X, Y),$$

that implies

$$d^\alpha_\infty\left(D_\Delta(X), D_\Delta(Y)\right) \leq 2^\alpha \left(d^d_{GH}(X, Y)\right)^\alpha.$$

We have by Corollary 2.49 that

$$d_\infty\left(D'_\Delta(X), D'_\Delta(Y)\right) \leq \beta \cdot d^\alpha_\infty\left(D_\Delta(X), D_\Delta(Y)\right).$$

Therefore,

$$d_\infty\left(D'_\Delta(X), D'_\Delta(Y)\right) \leq 2^\alpha \beta \left(d^d_{GH}(X, Y)\right)^\alpha.$$

□

We can extend the previous result if the function γ^{-1} is also (D, η)-Hölder continuous.

Lemma 2.51. *If γ^{-1} is (D, η)-Hölder continuous, then*

$$d_{GH}^{d}(X, Y) \leq 2^{\eta-1} D \cdot \left(d_{GH}^{d'}(X, Y)\right)^{\eta}. \tag{27}$$

Proof. Note that if $d' = \gamma \circ d$ then $d = \gamma^{-1} \circ d'$. By definition of distortion 2.19 we have for all correspondences $C : X \rightrightarrows Y$

$$\begin{aligned}
\operatorname{dis}^{d}(C) &= \sup\left\{\left|d(x, x') - d(y, y')\right| : (x, y), (x', y') \in C : X \rightrightarrows Y\right\} \\
&= \sup\left\{\left|(\gamma^{-1} \circ d')(x, x') - (\gamma^{-1} \circ d')(y, y')\right| : (x, y), (x', y') \in C : X \rightrightarrows Y\right\} \\
&\leq \sup\left\{D \cdot \left|d'(x, x') - d'(y, y')\right|^{\eta} : (x, y), (x', y') \in C : X \rightrightarrows Y\right\} \\
&\leq D \cdot \sup{}^{\eta}\left\{\left|d'(x, x') - d'(y, y')\right| : (x, y), (x', y') \in C : X \rightrightarrows Y\right\} = D \cdot \left(\operatorname{dis}^{d'}(C)\right)^{\eta}.
\end{aligned} \tag{28}$$

By definition of Gromov-Hausdorff distance, we have

$$d_{GH}(X, Y) = \frac{1}{2}\inf\left\{\operatorname{dis}(C) : C \text{ is a correspondence } X \rightrightarrows Y\right\}. \tag{29}$$

Note that the set $\{C : C \text{ is a correspondence } X \rightrightarrows Y\}$ is finite as X and Y are finite sets by definition of correspondence 2.18. Thus, the infimum of the Gromov-Hausdorff distance is attained in the set $\{\operatorname{dis}(C) : C \text{ is a correspondence } X \rightrightarrows Y\}$ and thus Equation (29) is equivalent to

$$d_{GH}(X, Y) = \frac{1}{2}\min\left\{\operatorname{dis}(C) : C \text{ is a correspondence } X \rightrightarrows Y\right\}. \tag{30}$$

Take C' the correspondence attaining the minimum in $d_{GH}^{d'}(X, Y)$, that is $d_{GH}^{d'}(X, Y) = \frac{1}{2}\operatorname{dis}^{d'}(C')$. We have by Equation (28) that

$$\operatorname{dis}^{d}(C') \leq D \cdot \left(\operatorname{dis}^{d'}(C')\right)^{\eta} = D \cdot \left(2 d_{GH}^{d'}(X, Y)\right)^{\eta} = 2^{\eta} D \left(d_{GH}^{d'}(X, Y)\right)^{\eta}. \tag{31}$$

Note that $d_{GH}^{d}(X, Y) \leq \frac{1}{2}\operatorname{dis}^{d}(C')$ by definition of (30). Using this identity and multiplying both sides of Equation (31) by $\frac{1}{2}$ we get

$$d_{GH}^{d}(X, Y) \leq \frac{1}{2}\operatorname{dis}^{d}(C') \leq 2^{\eta-1} D \left(d_{GH}^{d'}(X, Y)\right)^{\eta}. \tag{32}$$

□

Now we are ready to prove our stability theorem for $d'(x, y) = 1 - |\operatorname{Corr}(x, y)|$. Let us first define what is the correlation coefficient of two vectors of the same length.

Definition 2.52. Define the average of a vector $x = (x_1, \ldots, x_m) \in \mathbb{R}^m$ as

$$\bar{x} = \frac{1}{m}\sum_{i=1}^{m} x_i. \tag{33}$$

Define the variance of a vector $x = (x_1, \ldots, x_m) \in \mathbb{R}^m$ as

$$\operatorname{Var}(x) = \frac{1}{m}\sum_{i=1}^{m} (x_i - \bar{x})^2. \tag{34}$$

Define also the covariance between vectors $x = (x_1, \dots, x_m), y = (y_1, \dots, y_m) \in \mathbb{R}^m$ as

$$\mathrm{Cov}(x, y) = \frac{1}{m} \sum_{i=1}^{m} (x_i - \bar{x})(y_i - \bar{y}). \tag{35}$$

Then, the sample Pearson correlation coefficient between two finite vectors $x, y \in \mathbb{R}^m$ is

$$\mathrm{Corr}(x, y) = \frac{\mathrm{Cov}(x, y)}{\sqrt{\mathrm{Var}(x)\mathrm{Var}(y)}} = \frac{\sum_{i=1}^{m} (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{m} (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^{m} (y_i - \bar{y})^2}}. \tag{36}$$

In particular, the correlation coefficient is well defined for all vectors x, y such that $\mathrm{Var}(x) \neq 0 \neq \mathrm{Var}(y)$.

Lemma 2.53. *Let $\gamma : x \in [0,1] \to 1 - \sqrt{1 - x^2} \in [0,1]$. Then γ is $\left(\sqrt{2}, \frac{1}{2}\right)$-Hölder continuous.*

Proof.

$$\begin{aligned} |\gamma(x) - \gamma(y)| &= \left| \sqrt{1 - y^2} - \sqrt{1 - x^2} \right| \\ &\underbrace{\leq}_{\sqrt{x}\,(1,1/2)\text{-Hölder}} \left| 1 - y^2 - (1 - x^2) \right|^{1/2} = \left| x^2 - y^2 \right|^{1/2} \\ &\underbrace{\leq}_{x^2 \text{ is } 2-\text{Lipschitz in } [0,1]} (2 \cdot |x - y|)^{1/2} = \sqrt{2} \cdot |x - y|^{1/2}. \end{aligned}$$

□

Lemma 2.54. *Let $\gamma : x \in [0,1] \to 1 - \sqrt{1 - x^2} \in [0,1]$. Then $\gamma^{-1} : y \in [0,1] \to \sqrt{1 - (1 - y)^2} \in [0,1]$ is the inverse of γ.*

Proof. We have

$$\begin{aligned} \left(\gamma^{-1} \circ \gamma\right)(x) &= \gamma^{-1}\left(1 - \sqrt{1 - x^2}\right) \\ &= \sqrt{1 - \left(1 - \left(1 - \sqrt{1 - x^2}\right)\right)^2} \\ &= \sqrt{x^2} = x. \end{aligned}$$

□

Lemma 2.55. *Let $\gamma^{-1} : x \in [0,1] \to \sqrt{1 - (1 - x)^2} \in [0,1]$. Then γ^{-1} is $\left(\sqrt{2}, \frac{1}{2}\right)$-Hölder continuous.*

Proof.

$$\begin{aligned} \left|\gamma^{-1}(x) - \gamma^{-1}(y)\right| &= \left| \sqrt{1 - (1 - x)^2} - \sqrt{1 - (1 - y)^2} \right| \\ &\underbrace{\leq}_{\sqrt{x}\,(1,1/2)\text{-Hölder}} \left| 1 - (1 - x)^2 - \left(1 - (1 - y)^2\right) \right|^{1/2} \\ &= \left| (1 - y)^2 - (1 - x)^2 \right|^{1/2} \\ &\underbrace{\leq}_{x^2 \text{ is } 2-\text{Lipschitz in } [0,1]} (2 \cdot |(1 - y) - (1 - x)|)^{1/2} = \sqrt{2} \cdot |x - y|^{1/2}. \end{aligned}$$

□

Corollary 2.56. *Let be* $X, Y \subseteq \mathbb{R}^m$ *sets of vectors with* $\mathrm{Var(X)} \neq 0 \neq \mathrm{Var(Y)}$. *Let* $d'(x,y) = 1 - |\mathrm{Corr(x,y)}|$ *and* $d(x,y) = \sqrt{1 - \mathrm{Corr}^2(\mathrm{x},\mathrm{y})}$. *Let* $\gamma : x \in [0,1] \to 1 - \sqrt{1-x^2} \in [0,1]$. *Then,*

$$d_\infty\left(D'_\Delta(X), D'_\Delta(Y)\right) \leq 2 \cdot \left(d^d_{GH}(X,Y)\right)^{1/2} \leq 2 \cdot \left(d^{d'}_{GH}(X,Y)\right)^{1/4}.$$

Proof. Note that γ is strictly increasing, $\left(\sqrt{2}, \frac{1}{2}\right)$-Hölder continuous by Proposition 2.53 with inverse γ^{-1} also $\left(\sqrt{2}, \frac{1}{2}\right)$-Hölder continuous by Proposition 2.55.

Therefore, by Theorem 2.50, noting that $\beta = \max\left(\sqrt{2}, 1\right) = \sqrt{2}$, we have

$$d_\infty\left(D'_\Delta(X), D'_\Delta(Y)\right) \leq 2 \cdot \left(d^d_{GH}(X,Y)\right)^{1/2}.$$

Also, by Lemma 2.51 we have

$$d^d_{GH}(X,Y) \leq \left(d^{d'}_{GH}(X,Y)\right)^{1/2}.$$

Joining both equations, we get

$$d_\infty\left(D'_\Delta(X), D'_\Delta(Y)\right) \leq 2 \cdot \left(d^d_{GH}(X,Y)\right)^{1/2} \leq 2 \cdot \left(\left(d^{d'}_{GH}(X,Y)\right)^{1/2}\right)^{1/2} = 2 \cdot \left(d^{d'}_{GH}(X,Y)\right)^{1/4}.$$

□

2.2.2 Differentiability with symmetric functions

Our objective in this section is to prove that we can also differentiate persistence diagrams coming from arbitrary Vietoris-Rips sublevel set filtrations given a symmetric function. From now on, let $X, Y \subseteq \mathbb{R}^d$ be two finite sets of cardinality n with ambient space $\mathbb{R}^d$ equipped with its usual smooth structure and let d' be a symmetric function $d' : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ that is $\mathcal{C}^r(U)$ where U is an open set. As $|X| = |Y| = n$, we can see our sets X, Y simply as points $(p_1, \ldots, p_n) \in \mathbb{R}^{nd}$ where $\mathbb{R}^{nd}$ is also equipped with its usual smooth structure.

As in 2.1.5, we work with functions

$$B_p : \mathcal{M} = \mathbb{R}^{nd} \xrightarrow{F} \mathbb{R}^K \xrightarrow{\mathrm{Dgm}_{p,\Delta}} Bar, \tag{37}$$

where $K = \mathcal{P}(\{1, \ldots, n\}) \setminus \{\emptyset\}$ and $F(p) : \sigma \in K \to \max_{i,j \in \sigma} d'(p_i, p_j) \in \mathbb{R}^K$. From now on, we will see our filter functions $f \in \mathbb{R}^K$ as points $(f(\sigma_1), \ldots, f(\sigma_{2^n-1})) \in \mathbb{R}^{2^n-1}$ where $\sigma_1, \ldots, \sigma_{2^n-1}$ are the simplices of K in any order (we will not use this ordering explicitly so it is not relevant) equipped with the usual euclidean distance $\|\cdot\|_2$ in $\mathbb{R}^{2^n-1}$. Our objective is to find the requirements for F such that $\mathrm{Dgm}_{p,\Delta} \circ F$ is r-differentiable. The following two definitions are central to our work.

Definition 2.57. ([26, Definition 4.2]) Given a filter function $f \in \mathbb{R}^K$, the increasing order of its values induce a pre-order on the simplices of K. Two filter functions $f, g \in \mathbb{R}^K$ are said to be ordering equivalent, written $f \sim g$, if they induce the same pre-order on K.

Definition 2.58. ([26, Definition 4.3]) Given a filter function $f \in \mathbb{R}^K$ and a homology degree $0 \leq p \leq d$, a barcode template (P_p, U_p) is composed of a multiset P_p of pairs of simplices in K, together with a multiset U_p of simplices in K, such that:

$$\mathrm{Dgm}_{p,\Delta}\left(K^f_{sub}\right) = \left\{\left(f(\sigma), f(\sigma')\right)\right\}_{(\sigma,\sigma')\in P_p} \cup \left\{(f(\sigma), +\infty)\right\}_{\sigma\in U_p} \cup \Delta^\infty.$$

Here is the main theorem for local differentiability of filter functions.

Theorem 2.59. ([26, Theorem 4.7]) *Let $\mathcal{M}$ be a smooth finite-dimensional manifold without boundary (not necessarily $\mathbb{R}^{nd}$). Let $\theta \in \mathcal{M}$. Suppose the parametrisation $F : \mathcal{M} \to \mathbb{R}^K$ is of class $\mathcal{C}^r$, $r \geq 0$, on some open neighbourhood U of θ, and that $F(\theta) \sim F(\theta')$ for all $\theta' \in U$. Then B_p is r-differentiable.*

The following one is the global result.

Theorem 2.60. ([26, Theorem 4.9]) *Let $\mathcal{M}$ be a smooth finite-dimensional manifold without boundary (not necessarily $\mathbb{R}^{nd}$). Suppose the parametrization $F : \mathcal{M} \to \mathbb{R}^K$ is continuous over $\mathcal{M}$ and of class $\mathcal{C}^r$, $r \geq 0$, on some open subset U of $\mathcal{M}$. Then B_p is r-differentiable on the set $U \cap \bar{\mathcal{M}}$ where*

$$\bar{\mathcal{M}} = \left\{\theta \in \mathcal{M} \,|\, \exists \text{ open neighborhood } U_\theta \text{ of } \theta \text{ s.t. } F(\theta) \sim F(\theta') \text{ for all } \theta' \in U_\theta\right\},$$

which is generic (open and dense) in $\mathcal{M}$. In particular, if F is $\mathcal{C}^r$ on some generic subset of $\mathcal{M}$ in the first place, then so is B_p (on some possibly smaller generic subset).

Now suppose again that $\mathcal{M} = \mathbb{R}^{nd}$. Define the $\mathcal{C}^\infty$ projections

$$\pi_{i,j} : (p_1, \ldots, p_n) \in \mathbb{R}^{nd} \to (p_i, p_j) \in \mathbb{R}^d \times \mathbb{R}^d,$$

for all $1 \leq i, j \leq n$.

The following result characterises differentiability of persistence diagrams computed using Vietoris-Rips sublevel set filtrations with arbitrary symmetric functions d' that are $\mathcal{C}^r$ on an open subset U of $\mathbb{R}^{nd}$.

Proposition 2.61. *Let $d' : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ be a $\mathcal{C}^r$ symmetric function, $r \geq 0$, on an open set $U \subseteq \mathrm{Dom}(d)$. Let $P = (p_1, \ldots, p_n) \in \mathbb{R}^{nd}$. Suppose that*

1. *$\forall \{i, j\} \neq \{k, l\}$ where $i, j, k, l \in \{1, \ldots, n\}$, $d'(p_i, p_j) \neq d'(p_k, p_l)$;*
2. *$\forall i, j \in \{1, \ldots, n\}$, $P \in \pi^{-1}_{i,j}(U)$.*

Then $B_p = \mathrm{Dgm}_{p,\Delta} \circ \mathrm{F}$ is C^r at P, where $F : \mathbb{R}^{nd} \to \mathbb{R}^K$ is as in (37).

Proof. By Property 1 of P, the *distances* $d'(p_i, p_j)$ for $1 \leq i \neq j \leq n$ are strictly ordered. By the property that projections $\pi_{i,j}$ are $\mathcal{C}^\infty$ on all their domain and d' is $\mathcal{C}^r$ on U, with $\pi_{i,j}(P) \in U$ by Property 2 of P, then $d' \circ \pi_{i,j}$ is C^r in P. In particular $r \geq 0$, $d' \circ \pi_{i,j}$ is a continuous function on P. Therefore, there exists a neighbourhood U' of P where the order of the distances remains the same. Define now

$$\{\bar{v}(\sigma), \bar{w}(\sigma)\} = \arg\max_{i,j\in\sigma} d(p_i, p_j).$$

Therefore, for every $P' = (p'_1, \dots p'_n) \in U'$ we have $F(P')(\sigma) = d'\left(p'_{\bar{v}(\sigma)}, p'_{\bar{w}(\sigma)}\right)$ for all $\sigma \in K$. Take the non-empty open set

$$\bar{U} = U' \cap \bigcap_{i,j \in \{1,\dots,n\}}^{i \neq j} \pi_{i,j}^{-1}(U).$$

We have for all $P' \in \bar{U}$,

$$\begin{aligned} F(P') &= (f(\sigma_1), \dots, f(\sigma_{2^n-1})) \\ &= \left(\max_{i,j\in\sigma_1} d'(p'_i, p'_j), \dots, \max_{i,j\in\sigma_{2^n-1}} d'(p'_i, p'_j)\right) \\ &= \left(\max_{i,j\in\sigma_1} d'(\pi_{i,j}(P')), \dots, \max_{i,j\in\sigma_{2^n-1}} d'(\pi_{i,j}(P'))\right) \\ &= \left(\left(d' \circ \pi_{\bar{v}(\sigma_1),\bar{w}(\sigma_1)}\right)(P'), \dots, \left(d' \circ \pi_{\bar{v}(\sigma_{2^n-1}),\bar{w}(\sigma_{2^n-1})}\right)(P')\right), \end{aligned}$$

where, $\left(d' \circ \pi_{\bar{v}(\sigma_i),\bar{w}(\sigma_i)}\right)(P')$ is $\mathcal{C}^r$ in $\bar{U}$ by composition of C^r functions for all $1 \leq i \leq 2^n - 1$. Also, by definition of $\bar{U}$, the order of the distances is preserved (intersection with U') so we obtain that $F(P) \sim F(P')$ for all $P' \in \bar{U}$. Finally, the hypotheses of Theorem 2.59 hold and then B_p is r-differentiable in P. □

With this new generality, we can state the differentiability conditions for our function $d'(x, y) = 1 - |\mathrm{Corr}(x, y)|$.

Lemma 2.62. *Let*

$$\mathcal{Z} = \left\{P = (p_1, \dots, p_n) \in \mathbb{R}^{nd} : \mathit{Var}(p_i) \neq 0 \,\forall\, i \in \{1, \dots, n\}\right\}.$$

Then, $\mathcal{Z}$ is an open set of $\mathbb{R}^{nd}$.

Proof. Recall that, by Definition 2.52 the variance of $p_i = (p_{i,1}, \dots, p_{i,d}) \in \mathbb{R}^d$ is defined as

$$\mathrm{Var}(p_i) = \frac{1}{d}\sum_{i=1}^{d} (p_{i,j} - \bar{p}_i)^2.$$

Take a point $P = (p_1, \dots, p_n) \in \mathcal{Z}$. $\mathrm{Var}(\cdot)$ is continuous at each p_i because it is $\mathcal{C}^\infty$ on all $\mathbb{R}^d$. Also, $\mathrm{Var}(p_i) \neq 0$ for all $i \in \{1, \dots, n\}$ by definition of $\mathcal{Z}$. Therefore, there exists a neighbourhood $U \subseteq \mathbb{R}^{nd}$ of P such that for all $P' = (p'_1, \dots, p'_n) \in U$, $\mathrm{Var}(p'_i) \neq 0$ for all $i \in \{1, \dots, n\}$. Therefore, $U \subseteq \mathcal{Z}$ so $\mathcal{Z}$ is an open set in $\mathbb{R}^{nd}$. □

Corollary 2.63. *Let $\mathcal{Z} \subseteq \mathbb{R}^{nd}$ be as in Lemma 2.62. Let $d' : \mathbb{R}^d \times \mathbb{R}^d : (x, y) \mapsto 1 - |\mathrm{Corr}(x, y)| \in \mathbb{R}$. Then the function B_p given by the function $F : P \in \mathbb{R}^{nd} \to (\sigma \mapsto \max_{i,j\in\sigma} d'(p_i, p_j)) \in \mathbb{R}^K$ composed with $\mathrm{Dgm}_{p,\Delta}$, is $\mathcal{C}^\infty$ in the set*

$$\bar{\mathcal{Z}} = \left\{P = (p_1, \dots, p_n) \in \mathcal{Z} : \mathrm{Cov}(p_i, p_j) \neq 0 \,\forall\, i, j \in \{1, \dots, n\} \wedge \right.$$
$$\left. d'(p_i, p_j) \neq d'(p_k, p_l) \,\forall\, \{i, j\} \neq \{k, l\} \text{ where } i, j, k, l \in \{1, \dots, n\}\right\} \subseteq \mathcal{Z}.$$

Proof. By definition of correlation 2.52 we have

$$\text{Corr}(x,y) = \frac{\text{Cov}(x,y)}{\sqrt{\text{Var}(x)\text{Var}(y)}}.$$

Note that $1-y$ is $\mathcal{C}^\infty$ on all its domain whereas the correlation coefficient $\text{Corr}(x,y)$ is a rational function $\mathcal{C}^\infty$ on all the projections $\pi_{i,j}(P)$ with $P \in \bar{\mathcal{Z}}$ because variances of p_i and p_j are different from zero for all $i,j \in \{1,\dots,n\}$ by definition of $\mathcal{Z}$ in 2.62. The only problem for d' to be $\mathcal{C}^\infty$ in the projections $\pi_{i,j}(P)$ is given by the absolute value function $|\cdot|$ that is $\mathcal{C}^\infty$ everywhere except on zero. Therefore, $d'(x,y)$ is $\mathcal{C}^\infty$ whenever $\text{Corr}(x,y) \neq 0$ that is equivalent to have $\text{Cov}(x,y) \neq 0$. Note that for all $P \in \bar{\mathcal{Z}}$ we have $\text{Cov}(p_i,p_j) \neq 0$ for all $i,j \in \{1,\dots,n\}$. This is by assumption for $i \neq j$ and by noting that $\text{Cov}(p_i,p_i)$ is zero if and only if $\text{Var}(p_i) = 0$ because

$$\text{Cov}(p_i,p_i) = \frac{1}{n}\sum_{i=1}^{m}(x_i-\bar{x})^2 = \text{Var}(p_i),$$

that does not happen by hypothesis of P being in $\mathcal{Z}$. Then, d' is $\mathcal{C}^\infty$ on $\pi_{i,j}(P)$ for all $P \in \bar{\mathcal{Z}}$ and $i,j \in \{1,\dots,n\}$. Also, for all $P \in \bar{\mathcal{Z}}$ we have different $d'(p_i,p_j)$ by definition of $\bar{\mathcal{Z}}$. Therefore, the conditions of 2.61 hold and thus B_p is $\mathcal{C}^\infty$ on $\bar{\mathcal{Z}}$. □

Lemma 2.64. *$\bar{\mathcal{Z}} \subseteq \mathcal{Z}$ is an open set in $\mathbb{R}^{nd}$.*

Proof. Take $P = (p_1,\dots,p_n) \in \bar{\mathcal{Z}}$. We have that $\text{Cov}(p_i,p_j) \neq 0$ for all $i,j \in \{1,\dots,n\}$ as we saw in the proof of Corollary 2.63. Therefore, by continuity of the covariance $\text{Cov}(x,y)$ we have that there exists an open neighbourhood $U_1 \subseteq \mathbb{R}^{nd}$ of P such that $\text{Cov}(p_i,p_j) \neq 0$ for all $i,j \in \{1,\dots,n\}$. Also, $d'(x,y) = 1-|\text{Corr}(x,y)|$ is a $\mathcal{C}^\infty$ function in the projections $\pi_{i,j}(P)$ of points $P = \in \bar{\mathcal{Z}}$ as we saw in the proof of Corollary 2.63. Then, by continuity of $d' \circ \pi_{i,j}$ in P, there exists an open neighbourhood $U_2 \subseteq \mathbb{R}^{nd}$ of P such that $d'(p_i,p_j) \neq d'(p_k,p_l) \forall \{i,j\} \neq \{k,l\}$ where $i,j,k,l \in \{1,\dots,n\}$. Therefore, the set $\bar{U} = U_1 \cap U_2 \cap \mathcal{Z} \subseteq \mathcal{Z}$ is open as intersection of open sets in $\mathbb{R}^{nd}$ and satisfies $\bar{U} \subseteq \bar{\mathcal{Z}}$, so $\bar{\mathcal{Z}}$ is an open set of $\mathbb{R}^{nd}$. □

The next proposition gives us closed formulas to compute the gradients of our function B_p.

Proposition 2.65. ([26, Proposition 4.14]) *Let $\mathcal{M}$ be a smooth finite-dimensional manifold without boundary (not necessarily $\mathbb{R}^{nd}$). Let $F : \mathcal{M} \to \mathbb{R}^K$ continuous of class $\mathcal{C}^r$ on some open set $U \subseteq \mathcal{M}$ with $r \geq 1$. Given $\theta \in U \cap \bar{\mathcal{M}}$ with $\bar{\mathcal{M}}$ defined as in 2.60 and a barcode template (P_p, U_p) of $F(\theta)$, for any choice of ordering $(\sigma_1,\sigma_1'),\dots,(\sigma_m,\sigma_m'),\tau_1,\dots,\tau_n$ of (P_p,U_p), the map*

$$\bar{B}_p : \theta' \mapsto \left[\left(F(\theta')(\sigma_i), F(\theta')(\sigma_i')\right)_{i=1}^m, \left(F(\theta')(\tau_j)\right)_{j=1}^n\right],$$

is a local $\mathcal{C}^1$ lift of B_p around θ, and the corresponding differential for B_p at θ is:

$$d_{\theta,\bar{B}_p}B_p(\theta') = \left[\left(d_\theta F(\theta')(\sigma_i), d_\theta F(\theta')(\sigma_i')\right)_{i=1}^m, \left(d_\theta F(\theta')(\tau_j)\right)_{j=1}^n\right].$$

As $\bar{\mathcal{Z}}$ is an open set of the smooth manifold $\mathbb{R}^{nd}$, then $\bar{\mathcal{Z}}$ is a smooth submanifold without boundary with the inherited smooth structure. Now take $F' = F|_{\bar{\mathcal{Z}}} : \bar{\mathcal{Z}} \subseteq \mathbb{R}^{nd} \to \mathbb{R}^K$. With the previous Proposition 2.65 we can give explicit formulas for the differentials of B_p' given by the function F'.

Corollary 2.66. *Let* $d'(x,y) = 1 - |\mathrm{Corr}(x,y)|$, $F : P \in \mathbb{R}^{nd} \to (\sigma \mapsto \max_{i,j\in\sigma} d'(p_i, p_j)) \in \mathbb{R}^K$ *and* $F' = F|_{\bar{\mathcal{Z}}} : \bar{\mathcal{Z}} \subseteq \mathbb{R}^{nd} \to \mathbb{R}^K$. *Given* $P \in \bar{\mathcal{Z}}$ *and a barcode template* (P_p, U_p) *of* $F'(P)$, *for any choice of ordering* $(\sigma_1, \sigma_1'), \dots, (\sigma_m, \sigma_m')$, $\tau_1, \dots, \tau_n$ *of* (P_p, U_p), *the map*

$$\bar{B}_p : P' \mapsto \left[\left(F'(P')(\sigma_i), F'(P')(\sigma_i')\right)_{i=1}^m, \left(F'(P')(\tau_j)\right)_{j=1}^n \right],$$

is a local $\mathcal{C}^1$ *lift of* B_p *around* P, *and the corresponding differential for* B_p *at* P *is:*

$$d_{P,\bar{B}_p} B_p(P') = \left[\left(\left(\nabla_p d'\left(p_{\bar{v}(\sigma_i)}, p_{\bar{w}(\sigma_i)}\right)\right) \left(p'_{\bar{v}(\sigma_i)}, p'_{\bar{w}(\sigma_i)}\right), \left(\nabla_p d'\left(p_{\bar{v}(\sigma_i')}, p_{\bar{w}(\sigma_i')}\right)\right) \left(p'_{\bar{v}(\sigma_i')}, p'_{\bar{w}(\sigma_i')}\right) \right)_{i=1}^m, \right.$$
$$\left. \left(\left(\nabla_p d'\left(p_{\bar{v}(\tau_j)}, p_{\bar{w}(\tau_j)}\right)\right) \left(p'_{\bar{v}(\tau_j)}, p'_{\bar{w}(\tau_j)}\right) \right)_{j=1}^n \right],$$

where

$$\{\bar{v}(\sigma), \bar{w}(\sigma)\} = \mathrm{argmax}_{i,j\in\sigma} d'(p_i, p_j).$$

Proof. $\bar{\mathcal{Z}}$ is an open submanifold without boundary of $\mathbb{R}^{nd}$ because $\bar{\mathcal{Z}}$ is open in $\mathbb{R}^{nd}$ by Lemma 2.64. Also, by Corollary 2.63, points $P \in \bar{\mathcal{Z}}$ satisfy the conditions for points in Proposition 2.61, where we saw that there existed neighborhoods $\bar{U}_P$ of points P such that F was $\mathcal{C}^r$ in it. In our case, again by Corollary 2.63 and Proposition 2.61, $r = \infty$. As this happens for all $P \in \bar{\mathcal{Z}}$ then F' is $\mathcal{C}^\infty$ in all $\bar{\mathcal{Z}}$. Therefore we can apply Proposition 2.65 to obtain our local lifts $\bar{B}_p$ at a neighbourhood U_P of P. Note that also, this lift is also valid in the open set $\mathcal{U} = U_P \cap \bar{U}_P \cap \bar{\mathcal{Z}}$. By Proposition 2.61, in the neighborhood $\bar{U}_P$ we had

$$\begin{aligned} F(P') &= \left(\left(d' \circ \pi_{\bar{v}(\sigma_1), \bar{w}(\sigma_1)}\right)(P'), \dots, \left(d' \circ \pi_{\bar{v}(\sigma_{2^n-1}), \bar{w}(\sigma_{2^n-1})}\right)(P') \right) \\ &= \left(d'\left(p'_{\bar{v}(\sigma_1)}, p'_{\bar{w}(\sigma_1)}\right), \dots, d'\left(p'_{\bar{v}(\sigma_{2^n-1})}, p'_{\bar{w}(\sigma_{2^n-1})}\right) \right), \end{aligned}$$

for all $P' \in \mathcal{U} \subseteq \bar{U}_P$. Also, as $\mathcal{U} \subseteq \bar{\mathcal{Z}}$, we have $F'(P') = F(P)$ for all $P' \in \mathcal{U}$. By applying now the formula for differentials of Proposition 2.65 on this neighbourhood $\mathcal{U}$ we obtain directly the desired formula. □

3. Deep learning

3.1 Computational graphs

The objective of this work is to analyse the *dynamics* of neural networks in presence of input data and to use the insights obtained in improving neural networks during the training phase. A neural network is a particular type of a more general concept called computational graph.

Definition 3.1. ([11, Definition 1]) A computational graph G is a 6-tuple

$$\left(n, I, E, \left(u^i\right)_{i\in\{1,\dots,n\}\subseteq\mathbb{N}}, \left(d^i\right)_{i\in\{1,\dots,n\}\subseteq\mathbb{N}}, \left(f^i\right)_{i\in\{I+1,\dots,n\}\subseteq\mathbb{N}}\right), \tag{38}$$

where $\mathcal{G} = (\{1, \dots, n\}, E)$ is a directed graph and where

1. vertices $\{1, \dots, l\}$ are sources (the only ones) in the directed graph (vertices with indegree zero) called **input vertices** with l satisfying $1 \leq l < n$;
2. E is the set of **directed edges** of the computational graph and satisfies that $\forall (i,j) \in E, \quad i < j$;
3. each $u^i \in \mathbb{R}^{d^i}$, $i \in \{1, \dots, n\}$, is the **variable** with **dimension** d^i associated with vertex i in the graph;
4. each f^i is a function $f^i \colon \mathbb{R}^{\bar{d}^i} \to \mathbb{R}^{d^i}$ where $\bar{d}^i = \sum_{(j,i)\in E} d^j$ defined on all its domain for all $i \in \{l+1, \dots, n\}$ called the **local function** for vertex i in the graph.

Write $N_{\text{in}}(i) = \{j : (j,i) \in E\}$. By convention, we write α^i for the vector consisting of vectors u^j concatenated for $(j,i) \in E$ given a fixed $i \in \{l+1, \dots, n\}$, i.e., $\alpha^i = \left(u^{j_1}, \dots, u^{j_{|N_{\text{in}}(i)|}}\right) \in \mathbb{R}^{\bar{d}^i}$ where $j^i \in N_{\text{in}}(i)$ for all $i \in \{1, \dots, |N_{\text{in}}(i)|\}$ and where $j^s < j^t$ iff $s < t$. Note that by 1 and 2 we have for all $(i,j) \in E$ that $i \in \{1, \dots, n-1\}$ and $j \in \{l+1, \dots, n\}$. Also, 2 implies that the graph $\mathcal{G}$ is acyclic and topologically ordered. When we refer to the computational graph G as a (directed) graph, we will be talking about the induced graph $\mathcal{G}$ instead. We will abuse our notation unless it is required by the context to stress the difference between both concepts.

Computational graphs define a clear way to obtain values for each non-input vertex variable u^i given specific instances of the input vertex variables $u_e^1, \dots, u_e^l$. The algorithm is very simple: for each non-input vertex n_i, we can compute a possible value for u^i by evaluating $f(\alpha_i)$. Starting from the vertices connected with the input vertices and updating at each step the vertices pointed by the vertices with updated values, we end up obtaining values for all the vertices in the computational graph. The previous algorithm is called the **forward algorithm for computational graphs**, and it is defined formally in Algorithm 1.

Algorithm 1: The forward algorithm for computational graphs

input : A computational graph $\left(n, l, E, \left(u^i\right)_{i\in\{1,\dots,n\}\subseteq\mathbb{N}}, \left(d^i\right)_{i\in\{1,\dots,n\}\subseteq\mathbb{N}}, \left(f^i\right)_{i\in\{l+1,\dots,n\}\subseteq\mathbb{N}}\right)$ and values for the input vertices' variables $u_e^1 \in \mathbb{R}^{d^1}, \dots, u_e^l \in \mathbb{R}^{d^l}$.

output: The same computational graph with the variables $\left(u^i\right)_{i\in\{1,\dots,n\}\subseteq\mathbb{N}}$ updated.

for $i \leftarrow 1$ **to** l **do**
  $u^i = u_e^i$
end
for $i \leftarrow (l+1)$ **to** n **do**
  $u^i = f^i(\alpha^i)$
end

Definition 3.2. An evaluation of a computational graph G at the input $u_e^1, \dots, u_e^l$ is the indexed family of values $\left(u^i\right)_{i\in\{1,\dots,n\}\subseteq\mathbb{N}}$ obtained by applying the forward algorithm with input values $u_e^1, \dots, u_e^l$.

Remark 3.3. Denote by o the number of sink vertices (outdegree zero vertices) of a computational graph G and let $n_1^o, \dots, n_o^o$ be these vertices, ordered by their vertex numbers, with associated variables $u_1^o \in \mathbb{R}^{d_1^o}, \dots, u_o^o \in \mathbb{R}^{d_o^o}$. From now on, we denote these sink vertices by **output vertices**. Note that the forward algorithm defines a function $G \colon \mathbb{R}^{d^1} \times \cdots \times \mathbb{R}^{d^l} \to \mathbb{R}^{d_1^o} \times \cdots \times \mathbb{R}^{d_o^o}$ that sends input variable values $u_e^1, \dots, u_e^l$ to the variable values of the output vertices of the evaluation of these input variable values in G.

Definition 3.4. Let $G = \left(n, l, E, \left(u^i\right)_{i\in\{1,\dots,n\}\subseteq\mathbb{N}}, \left(d^i\right)_{i\in\{1,\dots,n\}\subseteq\mathbb{N}}, \left(f^i\right)_{i\in\{l+1,\dots,n\}\subseteq\mathbb{N}}\right)$ be a computational graph. The function of G is the function $G\colon \mathbb{R}^{d^1}\times\cdots\times\mathbb{R}^{d^l} \to \mathbb{R}^{d^o_1}\times\cdots\times\mathbb{R}^{d^o_o}$ defined in Remark 3.3. We do not distinguish between the computational graph and its function when it is clear from the context.

3.2 Neural networks

There are many ways to define neural networks. In fact, neural networks models are so rich that the previous definition is not enough to properly describe all of them. However, several neural networks architectures can be described with the previous framework.

Definition 3.5. A neural network $\mathcal{N}$ is a 4-tuple

$$\left(\mathcal{N}_G, \left(f^i_\theta\right)_{i\in\{l+1,\dots,n\}}, \left(m_i\right)_{i\in\{l+1,\dots,n\}}, \left(\theta_i\right)_{i\in\{l+1,\dots,n\}}\right) \tag{39}$$

such that

1. $\mathcal{N}_G$ is a computational graph with $d^i = 1$ for all i;
2. $m_i \in \mathbb{N}_{>0}$, $\theta_i \in \mathbb{R}^{m_i}$ and $f^i_\theta(x,\theta)\colon \mathbb{R}^{\bar{d}_i}\times\mathbb{R}^{m_i}\to\mathbb{R}$ such that $f^i = f^i_\theta|_{\theta=\theta_i}$;
3. f^i_θ is $\mathcal{C}^r$, $r\geq 1$, in $\mathbb{R}^{\bar{d}_i}\times\mathbb{R}^{m_i}$.

The variables θ in the functions f^i_θ are the **trainable parameters** and the values θ_i are the **current parameters** of the neural network. Sometimes, we denote the vertices (and its associated variables) by **neurons**. We use both terms interchangeably. We abuse our notation and we write the function of the computational graph $\mathcal{N}_G$ defined in 3.4 also as $\mathcal{N}\colon \mathbb{R}^{d^1}\times\cdots\times\mathbb{R}^{d^l}\to\mathbb{R}^{d^o_1}\times\cdots\times\mathbb{R}^{d^o_o}$. We denote by **Nets** the set of all neural networks. We denote by $\mathbf{Nets}(\mathbb{R}^n)$ the set $\{\mathcal{N}\colon \mathcal{N}\in\mathbf{Nets}$ such that $\mathcal{N}$ has n input vertices$\}$ and by $\mathbf{Nets}(\mathbb{R}^n,\mathbb{R}^o)$ and $\mathbf{CNets}(\mathbb{R}^n,\mathcal{Y})$ the sets $\{\mathcal{N}\colon \mathcal{N}\in\mathbf{Nets}(\mathbb{R}^n)$ such that $\mathcal{N}$ has o output vertices$\}$ and $\{(\mathcal{N},\tau) : \mathcal{N}\in\mathbf{Nets}(\mathbb{R}^n), \mathrm{Codom}(\tau)=\mathcal{Y}$ such that $\tau\circ\mathcal{N}\colon\mathbb{R}^n\to\mathcal{Y}$ is well defined$\}$ respectively.

Definition 3.6. The extended neural network function $\bar{\mathcal{N}}$ of a neural network $\mathcal{N}$ is the $\mathcal{C}^r$ function $\bar{\mathcal{N}}\colon \mathbb{R}^l\times(\mathbb{R}^{m_{l+1}}\times\cdots\times\mathbb{R}^{m_n})\to\mathbb{R}^n$ with $\bar{\mathcal{N}}(x_1,\dots,x_l,\theta'_1,\dots,\theta'_{n-l}) = \left(e^1,\dots,e^n\right)$ where $\left(e^i\right)_{i\in\{1,\dots,n\}}$ is the evaluation 3.2 of the computational graph $\mathcal{N}'_G$ on the input variable values $x_1,\dots,x_l$ where $\mathcal{N}'$ is the neural network $\mathcal{N}$ but with parameters given by the inputs $\left(\theta'_{i-l}\right)_{i\in\{l+1,\dots,n\}}$.

Remark 3.7. The function $\bar{\mathcal{N}}$ is $\mathcal{C}^r$ because each output value given by the forward algorithm is yielded by a composition of $\mathcal{C}^r$ functions f^i_θ or by the identity function for the output variable values associated to input vertices. Sometimes we also consider the function $\bar{\mathcal{N}}_o\colon \mathbb{R}^l\times(\mathbb{R}^{m_{l+1}}\times\cdots\times\mathbb{R}^{m_n})\to\mathbb{R}^o$ given by $\bar{\mathcal{N}}$ projected to the o output vertices of $\mathcal{N}_G$ ordered increasingly by vertex numbers.

During the whole project we work with the most simple type of neural network structure: the **multilayer perceptron**. This model is simple enough to make easily interpretable experiments while being powerful enough to adapt to almost any task. However, all of our constructs can be generalised easily for more involved networks.

Definition 3.8. A multilayer perceptron is a neural network $\mathcal{N}$ defined by the 8-tuple

$$\left(i, h, \left(s^j\right)_{j\in\{1,\dots,h\}}, o, E, \left(u^i\right)_{i\in\{1,\dots,i+h_n+o\}}, \left(f_\theta^j\right)_{j\in\{i+1,\dots,i+h_n+o\}}, \left(\theta_i\right)_{i\in\{i+1,\dots,i+h_n+o\}}\right), \tag{40}$$

where $h_n = \sum_{j=1}^{h} s^j$ if $h > 0$ and $h_n = 0$ otherwise and where

1. $i \in \mathbb{N}_{>0}$ is the number of input vertices in the computational graph;
2. $h \in \mathbb{N}$ is the number of **hidden layers** (sets of vertices that are not input nor output);
3. $s^j \in \mathbb{N}_{>0}$ is the number of vertices in each hidden layer. If $h = 0$ then $\{s^j\} = \emptyset$;
4. $o \in \mathbb{N}_{>0}$ is the number of output vertices. Recall that output vertices are vertices with outdegree zero;
5. E is the set of edges of the computational graph associated to the neural network. The set E must satisfy some properties that we will state below;
6. $\left(u^i\right)_{i\in\{1,\dots,i+h_n+o\}}$ are the variable values of the neural network computational graph and they satisfy $u^i \in \mathbb{R}$ for all i;
7. The functions f_θ^j are the functions of the neural network and have the form $f_\theta^j : \mathbb{R}^{\bar{d}^j} \times \mathbb{R}^{m_j} \to \mathbb{R}$ where $m_j = \bar{d}^j + 1$ and $\theta_j \in \mathbb{R}^{m_j}$ for all j. These functions must satisfy some properties that we will state below.

The computational graph $\mathcal{N}_G$ for the neural network is built as follows:

1. We add vertices $I = \{1, \dots, i\}$ and $O = \{i + h_n + 1, \dots, i + h_n + o\}$ for the input and output vertices, respectively.
2. In case that $h > 0$, for each hidden layer $1 \le j \le h$, we add vertices $H_j = \{i + h_j + 1, \dots, i + h_{j+1}\}$ where $h_j = \sum_{i=1}^{j-1} s_j$ if $j > 1$ and $h_0 = 0$.

Let $H = \bigcup_{j=1}^{h} H_j$ if $h > 0$ and $H = \emptyset$ otherwise. The set of vertices is then given by $V = I \cup H \cup O$. The set of edges $E = \{(i, j) \mid i < j\}$ between these vertices must satisfy

1. The output values O can only be pointed by edges coming from vertices in the (last) hidden layer H_h if $h > 0$ or coming from vertices in the input set I if $h = 0$.
2. The vertices of the j-hidden layer H_j can only be pointed by neurons in the $j-1$-hidden layer H_{j-1} for $j \ge 2$.
3. The vertices of the (first) 1-hidden layer H_1, can be pointed only by the input vertices I.

The functions $f_\theta^j : \mathbb{R}^{\bar{d}^j} \times \mathbb{R}^{m_j} \to \mathbb{R}$ must have the form

$$f_\theta^j\left(x_1, \dots, x_{\bar{d}^j}, \theta_{j,1}, \dots, \theta_{j,\bar{d}^j+1},\right) = \sigma\left(\sum_{k=1}^{\bar{d}^j} \theta_{j,k} x_k + \theta_{j,\bar{d}^j+1}\right), \tag{41}$$

where $\sigma : \mathbb{R} \to \mathbb{R}$ is a non-linear $\mathcal{C}^r$ function in $\mathbb{R}$ with $r \geq 1$.

With these constraints and defining $n = i + h_n + o$, the computational graph $\mathcal{N}_G$ of the neural network is given by the 6-tuple

$$\left(n, i, E, \left(u^i\right)_{i\in\{1,\dots,i+h_n+o\}}, (1)_{i\in\{1,\dots,n\}}, \left(f_\theta^j|_{\theta=\theta_j}\right)_{j\in\{i+1,\dots,n\}}\right), \tag{42}$$

and the multilayer perceptron is the neural network $\mathcal{N}$ given by

$$\left(\mathcal{N}_G, \left(f_\theta^j\right)_{j\in\{i+1,\dots,n\}}, \left(\bar{d}^j + 1\right)_{j\in\{i+1,\dots,n\}}, (\theta_j)_{j\in\{i+1,\dots,n\}}\right). \tag{43}$$

Neural networks are usually used to approximate functions that are unknown but from which we know some information. Usually this information comes in the form of a dataset.

Definition 3.9. A dataset is a non-empty finite set of pairs

$$\left\{(x_i, y_i) \in \mathcal{X} \times \mathcal{Y} \subseteq \mathbb{R}^n \times \mathbb{R}^t\right\}_{i\in\{1,\dots,m\}}.$$

We usually assume that there exists a (usually unknown) function $f : \mathcal{X} \to \mathcal{Y}$ such that $f(x_i) = y_i$ for all $i \in \{1, \dots, m\}$ and that is of our interest in some way. This is the function that we want to approximate with neural networks. Depending on the form of this function, we say that we solve a problem of classification or a problem of regression. Therefore, it is convenient to define two different types of tasks in deep learning.

Definition 3.10. A regression dataset is a dataset $\mathcal{D} \subseteq \mathcal{X} \times \mathcal{Y} \subseteq \mathbb{R}^n \times \mathbb{R}^t$ such that there exists a function $f_\mathcal{D} \colon \mathcal{X} \to \mathcal{Y}$ satisfying $f(x) = y$ for all $(x, y) \in \mathcal{D}$. A regression neural network for a regression dataset $\mathcal{D}$ is a neural network $\mathcal{N} \in \mathbf{Nets}(\mathbb{R}^n, \mathbb{R}^t)$.

Definition 3.11. A classification dataset is a dataset $\mathcal{D} \subseteq \mathcal{X} \times \mathcal{Y} \subseteq \mathbb{R}^n \times \mathbb{N}$ such that there exists a function $f_\mathcal{D} \colon \mathcal{X} \to \mathcal{Y}$ satisfying $f(x) = y$ for all $(x, y) \in \mathcal{D}$. A classification neural network for a classification dataset $\mathcal{D}$ is a tuple $(\mathcal{N}, \tau) \in \mathbf{CNets}(\mathbb{R}^n, \mathcal{Y})$.

A regression problem consists of finding a regression neural network $\mathcal{N}$ that *best* approximates a specific function $f_\mathcal{D}$ in a regression dataset $\mathcal{D}$. A classification problem consists of finding a classification neural network $(\mathcal{N}, \tau)$ such that $\tau \circ \mathcal{N}$ *best* approximates a specific function $f_\mathcal{D}$ in a classification dataset $\mathcal{D}$. The definition of a good approximation of the desired function $f_\mathcal{D}$ by a neural network $\mathcal{N}$ depends on the *metric* used to compare them. We denote this metric by loss function.

Definition 3.12. Let $\mathrm{Hom}_\mathcal{D}(\mathcal{X}, \mathcal{Y}) = \{f \in \mathrm{Hom}(\mathcal{X}, \mathcal{Y}) : f(x) = y \text{ for all } (x, y) \in \mathcal{D}\}$. A loss function $\mathcal{L}$ for a dataset $\mathcal{D} \subseteq \mathcal{X} \times \mathcal{Y} \subseteq \mathbb{R}^n \times \mathbb{R}^t$ is a function $\mathcal{L}(N, f, D) \colon \mathbf{Nets}(\mathbb{R}^n) \times \mathrm{Hom}_\mathcal{D}(\mathcal{X}, \mathcal{Y}) \times \mathcal{P}(\mathbb{R}^n \times \mathbb{R}^t) \to \mathbb{R}$. Note that the same function $\mathcal{L}$ can be a loss function for several datasets.

Definition 3.13. Let $\mathcal{D} \subseteq \mathcal{X} \times \mathcal{Y} \subseteq \mathbb{R}^n \times \mathbb{R}^t$ be a dataset, $\mathcal{L}$ a loss function for the dataset $\mathcal{D}$ and $f_\mathcal{D} \colon \mathcal{X} \to \mathcal{Y}$ the desired function to approximate. A targeted regression problem consists of the problem of finding

$$\arg\min_{\mathcal{N}\in\mathbf{Nets}(\mathbb{R}^n,\mathbb{R}^t)} \left(\mathcal{L}|_{f=f_\mathcal{D}, D=\mathcal{D}}(\mathcal{N})\right),$$

where $\mathcal{D}$ is a regression dataset. A targeted classification problem consists of the problem of finding

$$\arg\min_{(\mathcal{N},\tau)\in\mathbf{CNets}(\mathbb{R}^n,\mathcal{Y})} \left(\mathcal{L}|_{f=f_\mathcal{D}, D=\mathcal{D}}(\mathcal{N})\right),$$

where $\mathcal{D}$ is a classification dataset.

Functions $f_{\mathcal{D}}$ usually are not unique in regression and classification datasets, so it is not always clear how we find the specific function $f_{\mathcal{D}}$ that we want to approximate. Generally, a dataset is made from observations in the real world that respond to some natural laws whose models are unknown to us. In this kind of situations, the goal is to approximate specifically these models. MNIST [25] is one example of these kind of datasets. MNIST is a classification dataset that contains thousands of images of handwritten digits and the usual objective with this dataset is to learn a mathematical model $f_{\mathcal{D}}\colon \mathbb{R}^{28\times 28} \to \{0,\dots,9\}$ that is capable of identifying handwritten numbers in the range $\{0,\dots,9\}$ in **any** 28×28 pixels image containing them. In this case, there are some *unknown* natural laws related to vision that allow humans to discern between the different digits. However, with the previous loose definition, there may be several models $f_{\mathcal{D}}$ that do the task of modelling the vision system for this concrete task. This is the case in the majority of deep learning problems. The usual solution is to relax the problem of finding an specific function $f_{\mathcal{D}}$ to approximate and just require that the selected loss function depends only on the values of the dataset $\mathcal{D}$ and the neural network $\mathcal{N}$. Therefore, from now on we use loss functions for datasets $\mathcal{D}\subseteq\mathcal{X}\times\mathcal{Y}\subseteq\mathbb{R}^n\times\mathbb{R}^t$ depending only on the parameters N and D, i.e., in functions $\mathcal{L}(N,f,D)=\mathcal{L}(N,D)\colon \mathbf{Nets}\left(\mathbb{R}^n\right)\times\mathcal{P}\left(\mathbb{R}^n\times\mathbb{R}^t\right)\to\mathbb{R}$. With these new losses, we can define the general regression and classification problems.

Definition 3.14. Let $\mathcal{D}\subseteq\mathcal{X}\times\mathcal{Y}\subseteq\mathbb{R}^n\times\mathbb{R}^t$ be a dataset and $\mathcal{L}(N,D)\colon \mathbf{Nets}\left(\mathbb{R}^n\right)\times\mathcal{P}\left(\mathbb{R}^n\times\mathbb{R}^t\right)\to\mathbb{R}$ a loss function for the dataset $\mathcal{D}$. A regression problem consists of the problem of finding

$$\arg\min_{\mathcal{N}\in\mathbf{Nets}(\mathbb{R}^n)}\left(\mathcal{L}|_{D=\mathcal{D}}(\mathcal{N})\right),$$

where $\mathcal{D}$ is a regression dataset. A classification problem consists of the problem of finding

$$\arg\min_{(\mathcal{N},\tau)\in\mathbf{CNets}(\mathbb{R}^n,\mathcal{Y})}\left(\mathcal{L}|_{D=\mathcal{D}}(\mathcal{N})\right),$$

where $\mathcal{D}$ is a classification dataset.

The previous problems are too difficult to solve directly due to the enormous quantity of configurations we can use to build neural networks, as we saw in the previous sections. The usual approach is to select a first neural network $\mathcal{N}=\left(\mathcal{N}_G,\left(f_\theta^i\right)_{i\in\{l+1,\dots,n\}},\left(m_i\right)_{i\in\{l+1,\dots,n\}},\left(\theta_i\right)_{i\in\{l+1,\dots,n\}}\right)$ and then to select *optimal parameters* $\left(\theta_i\right)_{i\in\{l+1,\dots,n\}}$ for $\mathcal{N}$ by minimising the function $\bar{\mathcal{L}}\colon \mathbb{R}^{m_{l+1}}\times\cdots\times\mathbb{R}^{m_n}\to\mathbb{R}$ given by $\bar{\mathcal{L}}\left(\theta'_1,\dots,\theta'_{n-l}\right)=\mathcal{L}\left(\mathcal{N}',\mathcal{D}\right)$ where $\mathcal{N}'$ is the neural network $\mathcal{N}$ but with parameters $\left(\theta'_{i-l}\right)_{i\in\{l+1,\dots,n\}}$. Notice that when we fix a first neural network $\mathcal{N}$ and we try to obtain its *optimal* parameters $\left(\theta_i\right)$ we can give more precise loss function definitions depending on the *shape* of $\mathcal{N}$.

Definition 3.15. A loss function for a dataset $\mathcal{D}\subseteq\mathcal{X}\times\mathcal{Y}\subseteq\mathbb{R}^n\times\mathbb{R}^t$ and a neural network $\mathcal{N}=\left(\mathcal{N}_G,\left(f_\theta^i\right)_{i\in\{l+1,\dots,n\}},\left(m_i\right)_{i\in\{l+1,\dots,n\}},\left(\theta_i\right)_{i\in\{l+1,\dots,n\}}\right)\in\mathbf{Nets}\left(\mathbb{R}^n,\mathbb{R}^t\right)$ is a function

$$\mathcal{L}(\theta'_1,\dots,\theta'_{n-l},D)\colon \mathbb{R}^{m_{l+1}}\times\cdots\times\mathbb{R}^{m_n}\times\mathcal{P}\left(\mathbb{R}^n\times\mathbb{R}^t\right)\to\mathbb{R}.$$

Definition 3.16. Let $\mathcal{D}\subseteq\mathcal{X}\times\mathcal{Y}\subseteq\mathbb{R}^n\times\mathbb{R}^t$ be a dataset and $\mathcal{N}$ a neural network. An iterative process that obtains parameters $\left(\theta_i\right)_{i\in\{l+1,\dots,n\}}$ using a loss function for the dataset $\mathcal{D}$ and the neural network $\mathcal{N}$ as defined in 3.15 is called a training process of the neural network $\mathcal{N}$.

Training processes for losses $\mathcal{L}$ usually fix a a subset $\mathcal{D}'\subseteq\mathcal{D}$ (that can be the complete set $\mathcal{D}$) at each iteration and then perform a gradient descent step (or any other iterative optimisation procedure) in the function $\mathcal{L}|_{D=\mathcal{D}'}$, with the hope of minimising $\mathcal{L}|_{D=\mathcal{D}}$. Notice that we do not require

that iterative processes of training processes use the whole dataset $\mathcal{D}$ at each iteration. This is simply because some training algorithms work better this way, like Stochastic Gradient Descent or Adam. Computational graph functions are usually complex functions that induce complex losses with many local minimum in which training algorithms often get stuck before arriving to a global minimum. Choosing a good training algorithm is one of the most relevant decisions for machine learning practitioners to be successful when training neural models. For our experiments, we will fix Adam [24] to be our training algorithm. We will give more details in Section 4.

There are many loss functions and training methods to obtain the *optimal* values for parameters of a neural network $\mathcal{N}$. Most of them relies on being able to compute the partial derivatives of the extended neural network function $\bar{\mathcal{N}}$ defined in 3.6 restricted to the values given by the output vertices. To obtain its Jacobian in an specific point $x_1, \dots, x_l, \theta'_1, \dots, \theta'_{n-l}$ we apply the chain rule in an efficient algorithm called backpropagation, specified in Algorithm 2, with input a computational graph $\mathcal{N}'_G$ and values $x_1, \dots, x_l, \theta'_1, \dots, \theta'_{n-l}$ where $\mathcal{N}'_G$ is the computational graph derived from $\mathcal{N}_G$ by changing the functions $f^j_\theta|_{\theta=\theta_j}$ for the functions f^j_θ and by adding input vertices representing the parameters of each function pointing only to their respective function vertices. The backpropagation algorithm for computational graphs and its use to obtain partial derivatives is extensively explained in the notes [11] by Michael Collins.

Algorithm 2: The backpropagation algorithm for computational graphs

input : A computational graph $G = \left(n, l, E, \left(u^i\right)_{i\in\{1,\dots,n\}\subseteq\mathbb{N}}, \left(d^i\right)_{i\in\{1,\dots,n\}\subseteq\mathbb{N}}, \left(f^i\right)_{i\in\{l+1,\dots,n\}\subseteq\mathbb{N}}\right)$ and values for the input vertices' variables $u_e^1 \in \mathbb{R}^{d^1}, \dots, u_e^l \in \mathbb{R}^{d^l}$ such that $\frac{\partial f^i}{\partial u^j}$ is well defined for all $\mathrm{Dom}(f^i)$ for all $(i,j) \in E$.

output: For $j \in \{1, \dots, l\}$ and $s \in \{i : i \text{ is a sink vertex of } G\}$ output the Jacobians $P_s^j = \frac{\partial u^s}{\partial u^j}|_{u_e^1,\dots,u_e^l}$

$G \leftarrow$ Forward Algorithm(G, $u_e^1, \dots, u_e^l$)
$S \leftarrow \{\}$ `// List of sink vertices.`
for $j \leftarrow n$ **to** 1 **do**
  if *vertex j is a sink* **then**
    $S \leftarrow S \cup \{j\}$
    for $s \in S$ **do**
      if $s = j$ **then**
        $P_s^j \leftarrow I(d^j)$ where $I(d^j)$ is the identity matrix of dimension $d^j \times d^j$.
      else
        $P_s^j \leftarrow 0(d^s \times d^j)$ where $0(d^s \times d^j)$ is the zero matrix of dimension $d^s \times d^j$.
        $P_j^s \leftarrow 0(d^j \times d^s)$
      end
    end
  else
    Compute $J^{j\to i}\left(\alpha^i\right)$ where $J^{j\to i}\left(\alpha^i\right) = \frac{\partial f^i}{\partial u^j}\left(\alpha^i\right)$
    for $s \in S$ **do**
      $\underbrace{P_s^j}_{d^s\times d^j} = \sum_{i:(j,i)\in E} \underbrace{P_s^i}_{d^s\times d^i} \times \underbrace{J^{j\to i}\left(\alpha^i\right)}_{d^i\times d^j}$
    end
  end
end

When we perform a training process given a first neural network $\mathcal{N}$ and a dataset $\mathcal{D}$, we usually expect that the trained neural network $\mathcal{N}$ performs properly in presence of new data sampled from the same source. However, it is usual that, after training, the neural network have *memorised* data from the dataset used during training, performing properly there but generalising badly to unseen data. One extreme example of this behaviour are polynomial interpolations in regression problems, where points in the dataset are predicted without errors but predictions outside the dataset make no sense. To avoid that, it is usual to split the complete dataset $\mathcal{D}$ into two disjoints datasets $\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{test}} \subsetneq \mathcal{D}$ such that $\mathcal{D} = \mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{test}}$. The training is then performed in the dataset $\mathcal{D}_{\text{train}}$ and then the performance of the training is tested in the unseen dataset $\mathcal{D}_{\text{test}}$. There are several ways to test the performance of a trained network on an unseen dataset $\mathcal{D}_{\text{test}}$. The easiest one is to compute also the loss for the test set and interpret the result. However, loss functions are usually hard to interpret and other metrics are used. The accuracy of a network is one of the most used metrics to test if a neural network is generalising properly outside the training dataset $\mathcal{D}_{\text{train}}$ in classification problems. It measures the percentage of inputs classified correctly.

Definition 3.17. Let $\mathcal{D}$ a classification dataset. Let $(\mathcal{N}, \tau)$ a classification neural network for the

dataset $\mathcal{D}$. The accuracy of $\mathcal{N}$ in $\mathcal{D}$ is defined as

$$\frac{1}{|\mathcal{D}|}\left|\{(x,y)\in\mathcal{D}:(\tau\circ\mathcal{N})(x)=y\}\right|.$$

Selecting appropriate loss functions is crucial to achieve good accuracies. There are several classical loss functions used in classification and regression problems, like the categorical cross-entropy loss implemented in Tensorflow [27], derived from the cross-entropy measure defined in [16, Equation 3.51]. To use this loss function for a a classification dataset $\mathcal{D}\subseteq\mathcal{X}\times\mathcal{Y}\subseteq\mathbb{R}^n\times\mathbb{N}\subseteq\mathbb{R}^n\times\mathbb{R}$ and a classification neural network $(\mathcal{N},\tau)$, we require that $\mathcal{Y}=\{1,\dots,m\}$ for $1\leq m<+\infty$, that $\mathcal{N}\in\mathbf{Nets}(\mathbb{R}^n,\mathbb{R}^m)$ and that $\tau\colon x=(x_1,\dots,x_m)\in\mathbb{R}^m\to\arg\max_{i\in\{1,\dots,m\}}x_i\in\mathcal{Y}=\{1,\dots,m\}$.

Definition 3.18. The categorical cross-entropy loss function for the classification dataset $\mathcal{D}\subseteq\mathcal{X}\times\{1,\dots,m\}\subseteq\mathbb{R}^k\times\mathbb{R}$ and the neural network $\mathcal{N}=\left(\mathcal{N}_G,\left(f_\theta^i\right)_{i\in\{l+1,\dots,n\}},(m_i)_{i\in\{l+1,\dots,n\}},(\theta_i)_{i\in\{l+1,\dots,n\}}\right)\in\mathbf{Nets}\left(\mathbb{R}^k,\mathbb{R}^m\right)$ is the function $\mathrm{CCR}_{\mathcal{N}}\colon\mathbb{R}^{m_{l+1}}\times\cdots\times\mathbb{R}^{m_n}\times\mathcal{P}(\mathbb{R}^n\times\mathbb{R})\to\mathbb{R}$ defined as

$$\mathrm{CCR}_{\mathcal{N}}(\theta'_1,\dots,\theta'_{n-l},D)\mapsto-\sum_{(x,y)\in D}\mathbb{1}^{\lfloor y\rfloor}\cdot ln\left(\mathrm{SoftMax}(\bar{\mathcal{N}}_o(x,\theta'_1,\dots,\theta'_{n-l}))\right),$$

where $\bar{\mathcal{N}}_o$ is defined as in Remark 3.7, $ln:x=(x_1,\dots,x_m)\in\mathbb{R}^m\to(ln(x_1),\dots,ln(x_m))\in\mathbb{R}^m$, $\mathbb{1}^\alpha$, $\alpha\in\mathbb{N}$, is the vector of m elements such that $\mathbb{1}^\alpha_j=1$ if $\alpha=j$ and $\mathbb{1}^\alpha_j=0$ otherwise, $\cdot$ is the usual euclidean dot product and SoftMax$\colon\mathbb{R}^m\to\mathbb{R}^m$ is the function such that $\mathrm{SoftMax}(x)_i=e^{x_i}/\sum_{j=1}^m e^{x_j}$.

Many times loss functions by themselves are not enough to achieve high generalisation on neural networks trained with these losses. Also, it is usual that loss functions only take into account a small group of desired properties to optimise during the training. However, there are many desired neural network properties that, when taken into account during the training process, improve the generalisation of the resulting neural networks. To take into account these other properties, it is usual to add terms to loss functions that capture how well a neural network is performing on these properties. These terms added to the main loss function are called regularisation terms.

Definition 3.19. Let $\mathcal{L}$ a loss function defined as in one of the previous Definitions 3.12 or 3.15 for the dataset $\mathcal{D}$. A regularisation term $\mathcal{T}$ is a loss function $\mathcal{T}$ of the same type as $\mathcal{L}$ which we use together in a training procedure.

3.2.1 Network functional graph and equivalent networks

When we train neural networks, our ideal objective would be to learn the maximum quantity of information of as possible, avoiding *knowledge redundancy*. This can be translated to having vertices whose functions have different outputs for each input fed to the network. In this section, we will discuss how studying correlations between activations of computational graph variables of neural networks allows us to define some kind of redundancy degree metric in a given neural network $\mathcal{N}$ for a given dataset $\mathcal{D}$. To do that, we will use the concept introduced in [1] of **network functional graph**.

Definition 3.20. Let $\mathcal{D}=\{(x_i,y_i)\}_{i\in\{1,\dots,m\}}\subseteq\mathcal{X}\times\mathcal{Y}$ be a dataset and G a computational graph with $d^j=1$ for all $j\in\{l+1,\dots,n\}$ whose function satisfies $\mathcal{X}\subseteq\mathrm{Dom}(G)$. Let $v\in V(G)$ be a non-input vertex of the computational graph and let u^v be its associated variable. The activation vector of the vertex v in presence of $\mathcal{D}$ is defined as the vector $a_v=(a_{v,i})_{i\in\{1,\dots,m\}}\in\mathbb{R}^m$ such that $a_{v,i}=u^v$ after applying the forward algorithm 1 with input values defined by x_i.

Note that this definition can be generalised easily to arbitrary computational graphs. However, as we work only with neural networks, we will use this simpler definition. Note that, fixing a dataset $\mathcal{D}$ of cardinality m, we have for each non-input vertex v a vector $a_v \in \mathbb{R}^m$. This will allow us to compare neurons easily in terms of their activation vectors.

Definition 3.21. Let $\mathcal{D}$ be a dataset with $|\mathcal{D}| = m$ and G a computational graph with $d^j = 1$ for all $j \in \{l+1, \dots, n\}$. The network functional graph of G in presence of $\mathcal{D}$ is the undirected weighted complete graph $\mathcal{F}(G, \mathcal{D}) = (V(\mathcal{F}(G, \mathcal{D})), E(\mathcal{F}(G, \mathcal{D})), w_{\mathcal{F}})$ where

- $w_{\mathcal{F}}$ is a symmetric function, i.e., if $(x, y) \in \mathrm{Dom}(w_{\mathcal{F}})$ then $(y, x) \in \mathrm{Dom}(w_{\mathcal{F}})$ and $w(x, y) = w(y, x)$ for all $(x, y) \in \mathbb{R}^m \times \mathbb{R}^m$;
- $V(\mathcal{F}(G, \mathcal{D})) = \{a_i\}_{i \in \{l+1, \dots, n\}}$ are the activation vectors of the non-input vertices of G in presence of $\mathcal{D}$;
- $E(\mathcal{F}(G, \mathcal{D})) = \{(a, b) : a, b \in V(\mathcal{F}(G, \mathcal{D}))$ with $a \neq b$ and $(a, b) \in \mathrm{Dom}(w_{\mathcal{F}})\}$ such that the weight of each edge $(a, b) \in E$ is $w_{\mathcal{F}}(a, b)$.

From now on, we will use the symmetric function $d(x, y) = 1 - |\mathrm{Corr}(x, y)|$ where $\mathrm{Corr}(x, y)$ is defined as in 2.52.

Proposition 3.22. *Let $x, y \in \mathbb{R}^m$ such that $m \geq 1$ and $\mathrm{Var(x)} \neq 0 \neq \mathrm{Var(y)}$. Let $d(x, y) = 1 - |\mathrm{Corr(x, y)}|$. Then $d(x, y) = 0$ if and only if there exist $a, b \in \mathbb{R}$ with $a \neq 0$ such that $y = ax + b\mathbb{1}_m$ where $\mathbb{1}_m = (1, \dots, 1) \in \mathbb{R}^m$.*

To prove this proposition, we need the following results.

Lemma 3.23. ([37]) *Let $a, b \in \mathbb{R}$. Then $\mathrm{Var(ax + by)} = \mathrm{a^2Var(x) + b^2Var(y) + 2abCov(x, y)}$.*

Lemma 3.24. *Let $x, y \in \mathbb{R}^m$, $m \geq 1$, and $a \in \mathbb{R}$. There exists $b \in \mathbb{R}$ such that $y = ax + b\mathbb{1}_m$ if and only if $\mathrm{Var(y - ax)} = 0$.*

Proof. $\Longrightarrow$]: Note that

$$\overline{y - ax} = \frac{1}{m}\sum_{i=1}^{m} y_i - ax_i = \frac{1}{m}\sum_{i=1}^{m}(ax_i + b - ax_i) = b. \tag{44}$$

Therefore, we obtain

$$\mathrm{Var}(y - ax) = \frac{\sum_{i=1}^{m}(y_i - ax_i - \overline{y - ax})^2}{m} = \frac{\sum_{i=1}^{m}(ax_i + b - ax_i - b)^2}{m} = 0. \tag{45}$$

$\Longleftarrow$]: We have

$$0 = \mathrm{Var}(y - ax) = \frac{\sum_{i=1}^{m}(y_i - ax_i - \overline{y - ax})^2}{m} \implies 0 = \sum_{i=1}^{m}(y_i - ax_i - \overline{y - ax})^2. \tag{46}$$

As each of the terms of the sum is positive or zero and the sum is zero, then each term must be zero so we get:

$$0 = (y_i - ax_i - \overline{y - ax})^2 \implies 0 = y_i - ax_i - \overline{y - ax} \implies y_i = ax_i + \overline{y - ax} \quad \forall i \in \{1, \dots, m\}. \tag{47}$$

Thus, we get $y = ax + b\mathbb{1}_m$ with $b = \overline{y - ax}$, as we wanted to prove. □

Lemma 3.25. *Let* $x, y \in \mathbb{R}^m$ *such that* $\mathrm{Var}(x) \neq 0 \neq \mathrm{Var}(y)$. *Then* $|\mathrm{Corr}(x, y)| = 1$ *implies that there exists* $a \in \mathbb{R} \setminus \{0\}$ *such that* $\mathrm{Var}(y - ax) = 0$.

Proof. We know by lemma 3.23 that $\mathrm{Var}(y - ax) = \mathrm{Var}(y) + a^2\mathrm{Var}(x) - 2a\mathrm{Cov}(x, y)$ for any $a \in \mathbb{R}$. By definition we have

$$\mathrm{Corr}(x, y) = \frac{\mathrm{Cov}(x, y)}{\sqrt{\mathrm{Var}(x)\mathrm{Var}(y)}}, \tag{48}$$

Therefore, assuming $|\mathrm{Corr}(x, y)| = 1$ then $\mathrm{Cov}(x, y) \in \left\{\pm\sqrt{\mathrm{Var}(x)\mathrm{Var}(y)}\right\}$. Substituting we get

$$\mathrm{Var}(y - ax) = \mathrm{Var}(y) + a^2\mathrm{Var}(x) \pm 2a\sqrt{\mathrm{Var}(x)\mathrm{Var}(y)} = \left(\sqrt{\mathrm{Var}(y)} \pm a\sqrt{\mathrm{Var}(x)}\right)^2. \tag{49}$$

Solving the previous Expression (49) equal to zero we obtain

$$0 = \left(\sqrt{\mathrm{Var}(y)} \pm a\sqrt{\mathrm{Var}(x)}\right)^2 \implies 0 = \left(\sqrt{\mathrm{Var}(y)} \pm a\sqrt{\mathrm{Var}(x)}\right) \implies a = \pm\frac{\sqrt{\mathrm{Var}(y)}}{\sqrt{\mathrm{Var}(x)}} \neq 0, \tag{50}$$

that is well defined as $\mathrm{Var}(x) \neq 0$. Taking this a, we obtain $\mathrm{Var}(y - ax) = 0$, as we wanted to prove. □

Proof of Proposition 3.22. $\implies$]: By applying Lemma 3.25 first and Lemma 3.24 secondly, we obtain the desired result.

$\impliedby$]: Suppose that there exist $a, b \in \mathbb{R}$, with $a \neq 0$ such that $y = ax + b\mathbb{1}_m$. First note that

$$\bar{y} = \frac{1}{m}\sum_{i=1}^{m} y_i = \frac{1}{m}\sum_{i=1}^{m}(ax_i + b) = \frac{1}{m}\left(\sum_{i=1}^{m} ax_i + \sum_{i=1}^{m} b\right) = \frac{a}{m}\sum_{i=1}^{m} x_i + b = a\bar{x} + b. \tag{51}$$

Therefore, we have:

$$\begin{aligned}
\mathrm{Corr}(x, y) &= \frac{\sum_{i=1}^{m}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{m}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{m}(y_i - \bar{y})^2}} \\
&= \frac{\sum_{i=1}^{m}(x_i - \bar{x})(ax_i + b - (a\bar{x} + b))}{\sqrt{\sum_{i=1}^{m}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{m}(ax_i + b - (a\bar{x} + b))^2}} \\
&= \frac{\sum_{i=1}^{m}(x_i - \bar{x})(ax_i - a\bar{x})}{\sqrt{\sum_{i=1}^{m}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{m}(ax_i - a\bar{x})^2}} \\
&= \frac{a\sum_{i=1}^{m}(x_i - \bar{x})^2}{\sum_{i=1}^{m}\sqrt{a^2}(x_i - \bar{x})^2} \\
&= \frac{a}{|a|} = \pm 1.
\end{aligned} \tag{52}$$

□

Definition 3.26. We say that two classification neural networks $(\mathcal{N}_1, \tau_1)$ and $(\mathcal{N}_2, \tau_2)$ are equivalent in a classification dataset $\mathcal{D}$ if for all $(x, y) \in \mathcal{D}$ we have that $(\tau_1 \circ \mathcal{N}_1)(x) = (\tau_2 \circ \mathcal{N}_2)(x)$.

Take a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i\in\{1,\dots,m\}}$ and a neural network $\mathcal{N}$. Let $\mathcal{F}(\mathcal{N}_G, \mathcal{D})$ be the network functional graph of the neural network computational graph $\mathcal{N}_G$ in presence of $\mathcal{D}$. Take the subgraph $\mathcal{G}(\mathcal{N}_G, \mathcal{D}) = (V(\mathcal{G}(\mathcal{N}_G, \mathcal{D})), E(\mathcal{G}(\mathcal{N}_G, \mathcal{D}))) \subseteq \mathcal{F}(\mathcal{N}_G, \mathcal{D})$ such that $V(\mathcal{G}(\mathcal{N}_G, \mathcal{D})) = V(\mathcal{F}(\mathcal{N}_G, \mathcal{D}))$ and $(u, v) \in E(\mathcal{G}(\mathcal{N}_G, \mathcal{D}))$ if and only if $(u, v) \in E(\mathcal{F}(\mathcal{N}_G, \mathcal{D}))$ and $w_{\mathcal{F}}(u, v) = 0$ where $w_{\mathcal{F}}(u, v) = d(u, v) = 1 - |\mathrm{Corr}(u, v)|$.

Lemma 3.27. *For any connected component $C = \{a_i\}_{i\in I}$ of $\mathcal{G}(\mathcal{N}_G, \mathcal{D})$ we have that, for any $i, j \in I$, there exist $\alpha, \beta \in \mathbb{R}$ such that $a_j = \alpha a_i + \beta \mathbb{1}_m$.*

Proof. As C is a connected component of $\mathcal{G}(\mathcal{N}_G, \mathcal{D})$ there exists a path $a_{\sigma(1)}, a_{\sigma(2)}, ..., a_{\sigma(k)}$ where $a_{\sigma(i)} \in C$ and $a_{\sigma(1)} = a_i$, $a_{\sigma(k)} = a_j$. By definition of $\mathcal{G}(\mathcal{N}_G, \mathcal{D})$, any two connected vertices $a_{\sigma(s)}, a_{\sigma(s+1)}$ have $\left|Corr\left(a_{\sigma(s)}, a_{\sigma(s+1)}\right)\right| = 1$ for $s \in \{1, \dots, k-1\}$. Therefore, by Lemma 3.22 we have that there exists $\alpha_s, \beta_s \in \mathbb{R}$ such that $a_{\sigma(s+1)} = \alpha_s a_{\sigma(s)} + \beta \mathbb{1}_m$ for all $s \in \{1, \dots, k-1\}$. We claim that for all $t \in \{2, \dots, k\}$ there exists $\alpha, \beta \in \mathbb{R}$ such that $a_{\sigma(t)} = \alpha a_{\sigma(1)} + \beta \mathbb{1}_m$. We prove it by induction. The case $t = 2$ follows by applying the previous paragraph for $s = 1$. Suppose then for t. Again, by the previous paragraph, we have:

$$a_{\sigma(t+1)} = \alpha_t a_{\sigma(t)} + \beta_t \mathbb{1}_m \underbrace{=}_{I.H.} \alpha_t(\alpha a_{\sigma(1)} + \beta \mathbb{1}_m) + \beta_t \mathbb{1}_m = \alpha_t \alpha a_{\sigma(1)} + (\alpha_t \beta + \beta_t)\, \mathbb{1}_m. \tag{53}$$

□

The following proposition builds a neural network $\mathcal{N}_r$ from a neural network $\mathcal{N}$ that gives the same outputs for all the inputs defined by the dataset $\mathcal{D}$ but with less neurons. In particular, we keep only one vertex j_i for all the connected components C_i of $\mathcal{G}(\mathcal{N}_G, \mathcal{D}) \setminus \mathcal{O}$ where $\mathcal{O}$ is the set of sink vertices by substituting all the occurrences of the other neurons $\zeta \in C_i$ by a linear combination involving only j_i, as all the variable values u^ζ given by the forward algorithm for the inputs defined by the dataset $\mathcal{D}$ are linearly dependent for the same connected component C_i.

Proposition 3.28. *Let $\mathcal{D} = \{(x_i, y_i)\}_{i\in\{1,\dots,m\}} \subseteq \mathcal{X} \times \mathcal{Y}$ be a classification dataset and $(\mathcal{N}, \tau)$ a classification multilayer perceptron. Let $\{C_i\}_{i\in I}$ the connected components of $\mathcal{G}(\mathcal{N}_G, \mathcal{D}) \setminus \mathcal{O}$ where $\mathcal{O}$ is the set of sink (output) vertices of the neural network $\mathcal{N}$. Let $V_i = \{j : a_j \in C_i\}$ be the set of vertices in C_i for $i \in I$. There exists an equivalent classification neural network $(\mathcal{N}_r, \tau)$ induced by a directed graph G' with $V(G') \subseteq V(\mathcal{N}_G)$ containing the input and output vertices of $\mathcal{N}_G$ and at most one vertex for each V_i for $i \in I$.*

Proof. Take a connected component C_i with vertices V_i of $\mathcal{G}(\mathcal{N}_G, \mathcal{D}) \setminus \mathcal{O}$. Take $j_i = min(V_i)$. For $s \neq j_i$, $s \in V_i$, the values of u^s are not used in the computation of u^{j_i} by the forward algorithm 1. Suppose the contrary. Then, there exists $s \in V_i$ such that u^s is used in the computation of u^{j_i}. This means that there exist a path in the graph $\mathcal{N}_G$ starting at s and ending at j_i. Then, we have $s < j_i$. However, $j_i = min(V_i)$ by definition and $s \in V_i$ so $s \geq j_i$, arriving at a contradiction.

By Lemma 3.27 we have that there exist $\alpha_s^{j_i}, \beta_s^{j_i} \in \mathbb{R}$ such that $a_s = \alpha_s^{j_i} a_{j_i} + \beta_s^{j_i} \mathbb{1}_m$ for all $s \in V_i$. Let

$$G'_V = \bigcup_{i\in I} \{j_i\} \cup (V_{\text{in}} \cup V_{\text{out}}),$$

where V_{in} and V_{out} are the input and output vertices of the computational graph $\mathcal{N}_G$ respectively, and

$$G'_E = \bigcup_{i\in I} \bigcup_{s\in V_i\setminus\{j_i\}} \{(j_i, t) : (s, t) \in E(\mathcal{N}_G)\} \cup \left\{(a, b) \in E(\mathcal{N}_G) : a, b \in G'_V\right\}.$$

We define the graph $G' = (V(G'), E(G'))$ as the graph with $V(G') = \{s : s \in G'_V$ such that $\exists (s, t) \in G'_E\}$ and $E(G') = \{(s, t) \in G'_E$ and $s, t \in V(G')\}$.

Let $N'_{\text{in}}(v) = \{u' : (u', v) \in E(G')\}$ and $N_{\text{in}}(v) = \{u : (u, v) \in E(\mathcal{N}_G)\}$. Let $u'_1, \dots, u'_{\left|N'_{\text{in}}(v)\right|}$ and $u_1, \dots, u_{|N_{\text{in}}(v)|}$ be the ordered vertices of $N'_{\text{in}}(v)$ and $N_{\text{in}}(v)$ respectively, i.e., $u'_i < u'_j \iff i < j$

and $u_i < u_j \iff i < j$ for valid values of i, j. Let $\gamma_v \colon N_{\text{in}}(v) \to \{1, \dots, |N_{\text{in}}(v)|\}$ be a bijection assigning a vertex of $N_{\text{in}}(v)$ its position in the order, i.e., $u = u_{\gamma_v(u)}$ for $u \in N_{\text{in}}(v)$. Let $C \colon V(\mathcal{N}_G) \to \{C_i\}_{i \in I} \cup \{\emptyset\}$ be a function such that $C(v) = C_i$ if there exists $i \in I$ such that $v \in C_i$ and $C(v) = \emptyset$ otherwise. This function is well defined as having correlation equal to one in absolute value defines an equivalence relation (reflexivity and symmetry come directly from the definition while transitivity comes from Lemma 3.27). Let $\mathcal{S}(s) = \{t : t \in C(s) \text{ and } s = \min(C(s))\}$. Note that $\mathcal{S}(s) = \emptyset$ if $s \neq \min(C(s))$. Let I the number of input vertices of $\mathcal{N}_G$. For each vertex $v \in V(G') \setminus \{1, \dots, I\}$ let $g_\theta^v \colon \mathbb{R}^{\bar{\delta^v}} \times \mathbb{R}^{\bar{\delta^v}+1} \to \mathbb{R}$ where $\bar{\delta^v} = \sum_{(u,v) \in E(G')} d^u$ be the function defined as

$$g_\theta^v\left(x_1, \dots, x_{\bar{d^v}}, \theta'_{v,1}, \dots, \theta'_{v,\bar{\delta^v}+1},\right) = \sigma\left(\sum_{k=1}^{\bar{\delta^v}} \theta'_{v,k} x_k + \theta'_{v,\bar{\delta^v}+1}\right),$$

where

$$\theta'_{v,k} = \begin{cases} \theta_{\gamma_v(u'_k)} + \bar{\theta}'_{v,k}, & \text{if } u'_k \in N_{\text{in}}(v) \\ \bar{\theta}'_{v,k}, & \text{otherwise,} \end{cases} \quad \text{and} \quad \bar{\theta}'_{v,k} = \sum_{\zeta \in \mathcal{S}\left(u'_k\right) \cap N_{\text{in}}(v)} \theta_{\gamma_v(\zeta)} \alpha_\zeta^{u'_k},$$

and

$$\theta'_{v,\bar{\delta^v}+1} = \theta_{v,\bar{d^v}+1} + \sum_{k=1}^{\bar{\delta^v}} \sum_{\zeta \in \mathcal{S}\left(u'_k\right) \cap N_{\text{in}}(v)} \theta_{\gamma_v(\zeta)} \beta_\zeta^{u'_k}.$$

The previous graph G' and the functions and parameters defined before induce a neural network $\mathcal{N}_r$ by *reindexing* appropriately the vertices in such a way the original order of the vertices given by $\mathcal{N}_G$ is conserved. Now, to see that $(\mathcal{N}, \tau)$ and $(\mathcal{N}_r, \tau)$ are equivalent in the dataset $\mathcal{D}$ we need only to see that output vertices of $\mathcal{N}$ and $\mathcal{N}_r$ yield the same variable values given the same inputs defined by the elements in the dataset $\mathcal{D}$ after applying the forward algorithm. In particular, we will prove that vertices of G', $V(G') \subseteq V(\mathcal{N}_G)$, get the same variable values when applying the forward algorithm. Let $\sigma(1), \dots, \sigma(|V(G')|)$ the vertices of $V(G')$ ordered in ascending order. Instead of working directly with $\mathcal{N}_r$, we will apply forward algorithm 1 to the graph G' but iterating only on the vertices of G' in ascending order, this is, changing the lines of the forward algorithm `for` $i \leftarrow (I+1)$ `to` n `do` $u^i = f^i(\alpha^i)$ to `for` $i \leftarrow (I+1)$ `to` $|V(G')|$ `do` $u^{\sigma(i)} = f^{\sigma(i)}(\alpha^{\sigma(i)})$.

We use complete induction on the values $1, \dots, |V(G')|$. The case $i = 1$ is trivial as we have included in G' the input vertices so $u^{\sigma(1)} = u^1$ and both obtain the same values using the forward algorithm. Suppose now that it is satisfied for the indices $1, \dots, n-1$. We prove it for n. We have two cases, when $n \in \{1, \dots, I\}$, and then $u^{\sigma(n)} = u^n$, getting both $u^{\sigma(n)}$ and u^n the same values using the forward algorithm and when $n \in \{I+1, \dots, |V(G')|\}$. Assume the second case. Take the function

$$f_\theta^{\sigma(n)}\left(x_1, \dots, x_{d^{\bar{\sigma}(n)}}, \theta_{\sigma(n),1}, \dots, \theta_{\sigma(n),d^{\bar{\sigma}(n)}+1},\right) = \sigma\left(\sum_{k=1}^{d^{\bar{\sigma}(n)}} \theta_{\sigma(n),k} x_k + \theta_{\sigma(n),d^{\bar{\sigma}(n)}+1}\right).$$

Let $\Gamma : v \in V(\mathcal{N}_G) \mapsto \min(C(v)) \in V(\mathcal{N}_G)$. We can write the function $f_\theta^{\sigma(n)}$ defined in Equation 3.2.1 also as

$$\sigma\left(\theta_{\sigma(n),d^{\bar{\sigma}(n)}+1} + \sum_{\substack{k=1, \\ \Gamma\left(\gamma_{\sigma(n)}^{-1}(k)\right)=\gamma_{\sigma(n)}^{-1}(k)}}^{d^{\bar{\sigma}(n)}} \theta_{\sigma(n),k} x_k + \sum_{\substack{k=1, \\ \Gamma\left(\gamma_{\sigma(n)}^{-1}(k)\right)\neq\gamma_{\sigma(n)}^{-1}(k)}}^{d^{\bar{\sigma}(n)}} \theta_{\sigma(n),k} x_k\right). \tag{54}$$

Let $X_{\sigma(n)} = \left\{\Gamma\left(\gamma^{-1}_{\sigma(n)}(k)\right) : k \in \left\{1, \dots, d^{\bar{\sigma}(n)}\right\} \text{ and } \Gamma\left(\gamma^{-1}_{\sigma(n)}(k)\right) \neq \gamma^{-1}_{\sigma(n)}(k)\right\}$. Define the variables x'_ζ for $\zeta \in X_{\sigma(n)}$ and add them to the function $f_\theta^{\sigma(n)}$. Assume these variables get the variable values u^ζ in the forward algorithm. Therefore, for the dataset $\mathcal{D}$ we have that $x_k = \alpha^{\gamma^{-1}(k)}_{\Gamma\left(\gamma^{-1}_{\sigma(n)}(k)\right)} x'_{\Gamma\left(\gamma^{-1}_{\sigma(n)}(k)\right)} + \beta^{\gamma^{-1}(k)}_{\Gamma\left(\gamma^{-1}_{\sigma(n)}(k)\right)}$ for $k \in \left\{1, \dots, d^{\bar{\sigma}(n)}\right\}$ such that $\Gamma\left(\gamma^{-1}_{\sigma(n)}(k)\right) \neq \gamma^{-1}_{\sigma(n)}(k)$. Substituting in Equation 54 we get

$$\begin{aligned}\sigma\Bigg(\theta_{\sigma(n),d^{\bar{\sigma}(n)}+1} + \sum_{\substack{k=1,\\ \Gamma\left(\gamma^{-1}_{\sigma(n)}(k)\right)=\gamma^{-1}_{\sigma(n)}(k)}}^{d^{\bar{\sigma}(n)}} \theta_{\sigma(n),k} x_k + \\ \sum_{\substack{k=1,\\ \Gamma\left(\gamma^{-1}_{\sigma(n)}(k)\right)\neq\gamma^{-1}_{\sigma(n)}(k)}}^{d^{\bar{\sigma}(n)}} \theta_{\sigma(n),k}\left(\alpha^{\gamma^{-1}_{\sigma(n)}(k)}_{\Gamma\left(\gamma^{-1}_{\sigma(n)}(k)\right)} x'_{\Gamma\left(\gamma^{-1}_{\sigma(n)}(k)\right)} + \beta^{\gamma^{-1}_{\sigma(n)}(k)}_{\Gamma\left(\gamma^{-1}_{\sigma(n)}(k)\right)}\right)\Bigg).\end{aligned} \tag{55}$$

Developing and rearranging the expression inside the function σ in the previous Equation 55, removing the unused input variables of $f_\theta^{\sigma(n)}$ in Equation 55 and with a proper permutation of the position of the input variables we obtain the function $g_\theta^{\sigma(n)}$ defined before. As by inductive hypothesis all the variable values feed to the input variables of $g_\theta^{\sigma(n)}$ are the same as their equivalent variable values u^j in $\mathcal{N}_G$ after the application of the forward algorithm for all the inputs given by the dataset $\mathcal{D}$, we get that $f_\theta^{\sigma(n)}(\alpha^{\sigma(n)}) = g_\theta^{\sigma(n)}(\alpha'^{\sigma(n)})$ with $\alpha'^{\sigma(n)} = \left(u^{j_1}, \dots, u^{j_{|N'_{\text{in}}(\sigma(n))|}}\right)$ where $u^{j_1}, \dots, u^{j_{|N'_{\text{in}}(\sigma(n))|}}$ are the ordered (ascending) vertices of $N'_{\text{in}}(\sigma(n))$ as we wanted to prove. □

Note that the *reduced* neural network is not a multilayer perceptron, but a generalisation. The more neurons we remove to create $\mathcal{N}_r$, the more *redundant* is the network $\mathcal{N}$, as we get the exact same outputs for both networks. Proposition 3.28 can also be generalised to a broader set of neural network configurations, so comparing original and *reduced versions* of neural networks, $\mathcal{N}$ and $\mathcal{N}_r$ respectively, could be used to assess the quality of a trained neural network.

However, pairwise correlations in absolute value between neurons of a neural network will be different to one in most cases. This implies that when approximating linearly one neuron with another as before, we introduce an error term that modifies the subsequent activations in the forward algorithm, modifying ultimately the values of the output neurons. In general, we expect that little perturbations on the activation values of some neurons produce little perturbations on the output values as our neural network function is a composition of $\mathcal{C}^r$ functions. Therefore, if we substitute one neuron n_i by another neuron n_j as in the previous Proposition 3.28 such that their activations for a dataset $\mathcal{D}$, a_i and a_j respectively, are almost in a linear dependence (high correlation coefficient) then we expect that the output neuron variable values do not change too much. This is especially useful in classification, where we return one class for a range of output values of the computational graph where little perturbations in the output neurons usually do not change the label $y \in \mathcal{Y}$ predicted.

In particular if we want to substitute n_i by n_j, a function of our distance $1 - |\text{Corr}(x, y)|$ give us a lower bound on the distance between a_i and the best linear approximation of $a_i \in \mathbb{R}^m$ in terms of our activations $a_j \in \mathbb{R}^m$ with respect to their Euclidean distance.

To give the best linear approximation of a_i in terms of a_j we use the least squares method in a simple linear regression, as in [33]. In this method, we approximate a_i by the best linear model

depending on a_j according to the least squares method, i.e. obtaining $\hat{a}_i = (a_{i,1}, \dots, a_{i,m}) \in \mathbb{R}^m$ such that $\hat{a}_{i,k} = \alpha a_{j,k} + \beta$ for $k \in \{1, \dots, m\}$. Write

$$\text{SSR} = \sum_{k=1}^{m} (\hat{a}_{i,k} - \bar{a}_i)^2, \quad \text{SST} = \sum_{k=1}^{m} (a_{i,k} - \bar{a}_i)^2 \quad \text{and} \quad \text{SSE} = \sum_{k=1}^{m} (\hat{a}_{i,k} - a_{i,k})^2 = \|\hat{a}_i - a_i\|^2.$$

The coefficient of determination is $r^2 = \frac{\text{SSR}}{\text{SST}}$. It can be proved [33] that $\text{SST} = \text{SSR} + \text{SSE}$ and that $r^2 = \text{Corr}^2(a_i, a_j)$. From this, we can prove the following lemma.

Lemma 3.29. $\sqrt{\text{SSE}} = \|\hat{a}_i - a_i\| \geq \sqrt{\text{SST} \cdot (1 - |\text{Corr}(a_i, a_j)|)}$.

Proof. We have that the codomain of the correlation is $[-1, 1]$. Therefore, the codomain of the coefficient of determination is $[0, 1]$ (it is the square of the sample Pearson correlation). Also, we know that $\sqrt{x} \geq x$ for $0 \leq x \leq 1$. All together, this implies that $|\text{Corr}(a_i, a_j)| = \sqrt{r^2} \geq r^2$. Therefore, we get

$$\frac{\text{SSR}}{\text{SST}} = \frac{\text{SST} - \text{SSE}}{\text{SST}} = 1 - \frac{\text{SSE}}{\text{SST}} = r^2 \leq |\text{Corr}(a_i, a_j)| \iff 1 - |\text{Corr}(a_i, a_j)| \leq \frac{\text{SSE}}{\text{SST}} \iff$$
$$\text{SST}\,(1 - |\text{Corr}(a_i, a_j)|) \leq \text{SSE} \iff \sqrt{\text{SST}\,(1 - |\text{Corr}(a_i, a_j)|)} \leq \sqrt{\text{SSE}} = \|\hat{a}_i - a_i\|.$$

□

This directly implies the following corollary.

Corollary 3.30. *There exists* $k \in \{1, ..., m\}$ *such that* $|a_{i,k} - \hat{a}_{i,k}| \geq \sqrt{\frac{SST}{m} \cdot (1 - |Corr(a_i, a_j)|)}$.

Proof. Suppose not. Then we have $|a_{i,k} - \hat{a}_{i,k}| < \sqrt{\frac{\text{SST}}{m} \cdot (1 - |\text{Corr}(a_i, a_j)|)}$ for all $k \in \{1, ..., m\}$. Then,

$$\|\hat{a}_i - a_i\| = \sqrt{\sum_{k=1}^{m} (a_{i,k} - \hat{a}_{i,k})^2} = \sqrt{\sum_{k=1}^{m} |a_{i,k} - \hat{a}_{i,k}|^2} < \sqrt{\sum_{k=1}^{m} \left(\sqrt{\frac{\text{SST}}{m} \cdot (1 - |\text{Corr}(a_i, a_j)|)} \right)^2}$$
$$= \sqrt{\text{SST} \cdot (1 - |\text{Corr}(a_i, a_j)|)},$$

that is a contradiction with the previous Lemma 3.29. □

The previous Corollary 3.30 gives a lower bound on the noise (perturbation) added when changing the neuron activations a_i by the best linear approximation of a_i, $\hat{a}_i$, using a_j. This lower bound is tightly connected with the coefficient correlation: the highest it is, the lower bound on the error produced.

Therefore, from now on, our heuristic will be that, the higher the correlations between the neurons, the more probability of having redundant neurons in our neural network. Vietoris-Rips filtrations and their persistence diagrams keep track of these correlations between neurons. In particular, in dimension zero, deaths of points in persistence diagrams indicate merging of connected components in the network functional graph by adding at least one edge with weight equal to the death value. These weights, as defined before, are the function values $1 - |\text{Corr}(a_i, a_j)|$. Note that $1 - d$, where d is a death value, gives the absolute value of the correlation between two unspecified neurons. Therefore, the higher the values $1 - d$ are, the higher the probability of having redundant neurons. We summarise this information in the following score.

Definition 3.31. Let $d'(x,y) = 1 - |\text{Corr}(x,y)|$. The topological redundancy of a neural network $\mathcal{N}$ with respect to the dataset $\mathcal{D} = \{(x_i, y_i)\}_{i\in\{1,\dots,m\}}$ is

$$\mathcal{R}_0(\mathcal{N}, \mathcal{D}) = \sum_{(0,d)\in \text{Dgm}_0(VR_{d'}(V(G')))} 1 - d, \tag{56}$$

where G' is the network functional graph of the neural network $\mathcal{N}$ with dataset $\mathcal{D}$ and symmetric function $1 - |\text{Corr}(x,y)|$.

4. Experimental results

4.1 Regularisation terms

Our objective in this section is to find regularisation terms $\mathcal{T}$ that improve the neural network obtained from a regular classification training procedure using the categorical cross-entropy loss function $\mathcal{L}$ defined in 3.18. For our experiments, we use two different regularisation terms given a fixed neural network $\mathcal{N} = \left(\mathcal{N}_G, \left(f_\theta^i\right)_{i\in\{l+1,\dots,n\}}, (m_i)_{i\in\{l+1,\dots,n\}}, (\theta_i)_{i\in\{l+1,\dots,n\}}\right)$. Recall that we denote by $\mathcal{F}(\mathcal{N}, D)$ the network functional graph of a neural network $\mathcal{N}$ given a symmetric function $d'(x,y)$, in our case $1 - |\text{Corr}(x,y)|$. The regularisation terms are

1. The standard deviation of dimension zero deaths in a network functional graph. Let

$$\text{Dgm}_0^{<\infty}(\mathcal{N}, D) = \{(b,d) \in \text{Dgm}_0(VR_{d'}(V(\mathcal{F}(\mathcal{N}, D)))) : d < +\infty\},$$

and denote its cardinality by $C = \left|\text{Dgm}_0^{<\infty}(\mathcal{N}, D)\right|$. Give any order of the points $(b,d) \in \text{Dgm}_0^{<\infty}(\mathcal{N}, D)$ by decreasing value $d - b$, i.e., $\text{Dgm}_0^{<\infty}(\mathcal{N}, D) = \{(b_1, d_1), \dots, (b_C, d_C)\}$ such that $d_i - b_i > d_j - b_j$ if and only if $i < j$ for $i, j \in \{1, \dots, C\}$. Finally, fix $S \leq C$ and let $\text{Dgm}(\mathcal{N}, D) = \{(b_1, d_1), \dots, (b_S, d_S)\}$. The standard deviation of dimension zero deaths in a network functional graph is a function $\mathcal{T}_{\text{std}} \colon \mathbb{R}^{m_{l+1}} \times \cdots \times \mathbb{R}^{m_n} \times \mathcal{P}(\mathbb{R}^n \times \mathbb{R}) \to \mathbb{R}$ defined as

$$\mathcal{T}_{\text{std}}(\theta'_1, \dots, \theta'_{n-l}, D) = \sqrt{\sum_{(0,d)\in \text{Dgm}(\mathcal{N}', D)} \frac{d - \bar{d}}{|\text{Dgm}(\mathcal{N}', D)|}}, \text{ with } \bar{d} = \frac{1}{|\text{Dgm}(\mathcal{N}', D)|} \sum_{\substack{(0,d)\in \\ \text{Dgm}(\mathcal{N}', D)}} d, \tag{57}$$

where $\mathcal{N}' = \left(\mathcal{N}_G, \left(f_\theta^i\right)_{i\in\{l+1,\dots,n\}}, (m_i)_{i\in\{l+1,\dots,n\}}, \left(\theta'_{i-l}\right)_{i\in\{l+1,\dots,n\}}\right)$. We use this regularisation term due to it was shown in [1] that there existed a high correlation between the standard deviation and averages of deaths of Vietoris-Rips persistence diagrams of dimension zero computed with the symmetric function $1 - |\text{Corr}(x,y)|$ and the generalisation gap. In particular, we observe in [1, Figure 4.3] that the lower the standard deviation is, the lower (the better) the generalisation gap is.

2. The sampled topological redundancy. It is a function $\mathcal{T}_{\text{red}} \colon \mathbb{R}^{m_{l+1}} \times \cdots \times \mathbb{R}^{m_n} \times \mathcal{P}(\mathbb{R}^n \times \mathbb{R}) \to \mathbb{R}$ defined as

$$\mathcal{T}_{\text{red}}(\theta'_1, \dots, \theta'_{n-l}, D) = \sum_{\substack{(0,d)\in \\ \text{Dgm}(\mathcal{N}', D)}} 1 - d, \tag{58}$$

Let $\sigma(1), \dots, \sigma(|V(\mathcal{F}(\mathcal{N}, D))|)$ any fixed order of the vertices whose activations are in the network functional graph $\mathcal{F}(\mathcal{N}, D)$. Let $a_{\sigma(i)}$ their associated activation vectors. Note that $\mathcal{F}(\mathcal{N}', D)$ contains activation vectors for the same set of vertices of $\mathcal{N}_G$ for any neural network $\mathcal{N}'$ coming from the neural network $\mathcal{N}$ but with different parameters. We have regularisation terms that factorise as

$$\mathcal{T}: \mathbb{R}^{m_{l+1}} \times \cdots \times \mathbb{R}^{m_n} \times \mathcal{P}(\mathbb{R}^n \times \mathbb{R}) \xrightarrow{\mathcal{G}} \mathbb{R}^{md} \underbrace{\xrightarrow{F} \mathbb{R}^K \xrightarrow{\mathrm{Dgm}_{0,\Delta}} \mathbf{Bar}}_{B_0} \xrightarrow{\mathcal{O}} \mathbb{R},$$

where $m = |V(\mathcal{F}(\mathcal{N}, D))|$, $d = |\mathcal{D}_{\text{train}}|$, $\mathcal{G}$ is the function that builds the network functional graph $\mathcal{F}(\mathcal{N}', D)$ and outputs $(a_{\sigma(1)}, \dots a_{\sigma(m)}) \in \mathbb{R}^{md}$, F is the Vietoris-Rips filter function given by $d' = 1 - |\mathrm{Corr}(x, y)|$ as in 2.2.2 and $\mathcal{O}$ is a function that depends on the regularisation term.

Fix a set $D' \subseteq \mathcal{P}(\mathbb{R}^n \times \mathbb{R})$. We want to prove that $\mathcal{T}_{\text{std}}|_{D=D'}$ and $\mathcal{T}_{\text{red}}|_{D=D'}$ are $\mathcal{C}^r$ functions with $r \geq 1$. Assume B_0 is ∞-differentiable. Under this assumption, if we prove that $\mathcal{G}|_{D=D'}$ is $\mathcal{C}^r$ with $r \geq 1$ and $\mathcal{O}$ is r-differentiable with $r \geq 1$ then, by the chain rule and by Proposition 2.27, we get that $\mathcal{T}|_{D=D'}$ is $\mathcal{C}^r$ with $r \geq 1$. First note that $\mathcal{G}$ is automatically $\mathcal{C}^r$ with $r \geq 1$ because its output is the concatenation of $\mathcal{C}^r$, $r \geq 1$, extended neural network functions $\bar{\mathcal{N}}(x, \theta'_1, \dots, \theta'_{n-l})|_{x=x'}$ projected to the output values given by the indices associated with the vertices in the graph G' defined in 1 for $(x', y') \in D'$.

Now let $\mathrm{UD}_S = \{\mathcal{D} \in \mathbf{Bar}_\Delta : d_2 - b_2 \neq d_1 - b_1 \, \forall (b_1, d_1), (b_2, d_2) \in \mathcal{D} \setminus \Delta^\infty \text{ and } |\mathcal{D} \setminus \Delta^\infty| \geq S\}$. For any $\mathcal{D} \in \mathrm{UD}$, let $\mathcal{D}_S$ the set of points $(b, d) \in \mathcal{D} \setminus \Delta^\infty$ with the S highest $d - b$ values. Note that the set is well defined because $\mathcal{D} \in \mathrm{UD}_S$. Note that for the regularisation terms, if $B_0 \circ \mathcal{G} \in \mathrm{UD}_S$ the functions $\mathcal{O}_{\text{std}}$ and $\mathcal{O}_{\text{red}}$ are

$$\mathcal{O}_{\text{std}}(D) = \sqrt{\sum_{(0,d) \in D_S} \frac{d - \bar{d}}{|D_s|}} \text{ and } \mathcal{O}_{\text{red}}(D) = \sum_{(0,d) \in D_S} 1 - d,$$

respectively.

Take now $D \in \mathrm{UD}_S$, arbitrary $m, n \in \mathbb{Z}_{\geq 0}$ and a vector $\bar{D} = (b_1, d_1, \dots, b_m, d_m, b'_1, \dots, b'_l) \in \mathbb{R}^{2m} \times \mathbb{R}^n$ such that $Q_{m,n}(\bar{D}) = D$. Let $\sigma_b(1), \sigma_d(1), \dots \sigma_b(S), \sigma_d(S)$ the indices of the vector $\bar{D}$ corresponding to the points in $D_S = \{(b_1, d_1), \dots, (b_S, d_S)\}$ where $d_i - b_i > d_j - b_j$ for $i < j$ for all $i, j \in \{1, \dots, S\}$. Note also that for all $(b, d) \in D \setminus \Delta^\infty$ we have $d - b > 0$ and for all the points $(b, d) \in \Delta^\infty$ we have $d - b = 0$. By the strict order for points in $D \setminus \Delta^\infty$ and the previous property there exists a neighbourhood $\mathcal{U}_{\bar{D}}$ such that for all $\bar{D}' \in \mathcal{U}_{\bar{D}}$ $(Q_{m,n}(\bar{D}'))_S$ is well defined and the indices of the vector $\bar{D}'$ corresponding to the points in $(Q_{m,n}(\bar{D}'))_S = \{(b'_1, d'_1), \dots, (b'_S, d'_S)\}$ are the same as the ones for $\bar{D}$, i.e., $\sigma_b(1), \sigma_d(1), \dots \sigma_b(S), \sigma_d(S)$. Therefore, the functions $\mathcal{O}_{\text{std}} \circ Q_{m,n}$, $\mathcal{O}_{\text{red}} \circ Q_{m,n}$ are defined as

$$\mathcal{O}_{\text{std}} \circ Q_{m,n}(b_1, d_1, \dots, b_m, d_m, b'_1, \dots, b'_l) = \sqrt{\sum_{i=1}^{S} \frac{d_{\sigma_d(i)} - \bar{d}}{S}} \text{ with } \bar{d} = \frac{1}{S} \sum_{i=1}^{S} d_{\sigma_d(i)}$$

and

$$\mathcal{O}_{\text{std}} \circ Q_{m,n}(b_1, d_1, \dots, b_m, d_m, b'_1, \dots, b'_l) = \sum_{i=1}^{S} 1 - d_{\sigma_d(i)},$$

for all $\bar{D}' \in \mathcal{U}_{\bar{D}}$. These functions are $\mathcal{C}^r$ with $r \geq 1$ so $\mathcal{O}_{\text{std}}$ and $\mathcal{O}_{\text{red}}$ are r-differentiable with $r \geq 1$ and thus $\mathcal{T}_{\text{red}}$ and $\mathcal{T}_{\text{std}}$ are $\mathcal{C}^r$ whenever the proposed assumptions hold.

In our case we only add the regularisation terms whenever they can be computed, this is, given values $\theta'_1, \dots, \theta'_{n-l}$, D', and $S \in \mathbb{N}$, the value $\mathcal{G}|_{D=D'}(\theta'_1, \dots, \theta'_{n-l})$ is in the set $\bar{\mathcal{Z}}$ defined in Corollary 2.63 and the value $B_0 \circ \mathcal{G}|_{D=D'}$ is in the set UD_S defined before. In practice, this has happened every time during our experiments due to the difficulty in getting correlations between activation vectors that are pairwise equal or that have covariance equal to zero. This also means that we can fix our S to have a value less or equal to $|\mathcal{N}_G| - l - 1$ where l is the number of input vertices of $\mathcal{N}_G$. This is true because for all D' we have $|\mathcal{F}(\mathcal{N}, D')| - 1 = |\mathcal{N}_G| - l - 1$ and because if we have different pairwise distances for all the activation vectors then we have that for dimension zero persistence diagrams we will have at least $|\mathcal{F}(\mathcal{N}, D')| - 1$ points as at most two activation values will be connected in the simplicial complex generated by the Vietoris-Rips sublevel set filtration at distance zero. In our case, we selected $S = |\mathcal{N}_G| - l - 10$ due to we started the experiments before developing the theory and we did not know how was the structure of the set $\bar{\mathcal{Z}}$.

4.2 Dropout

Dropout is a regularisation technique introduced in [35] that is used during a training procedure to improve the generalisation capacity of a neural network. It consists of ignoring the output of a certain set of vertices selected randomly among the vertices of a neural network in each step of a training procedure. Note that the set of discarded vertices is different in each training iteration. In particular, we drop the edges coming out from the ignored vertices during the feedforward and backpropagation algorithms. In our case, we set a constant percentage to each vertex that fixes the probability of this neuron to be ignored during the training iteration.

4.3 Density functions of persistence diagrams

Persistence diagrams coming from dimension zero Vietoris-Rips persistence diagrams usually have the same birth for all their points. This implies that a direct plot of the persistence diagram is not usually useful and the statistics about the distribution of the deaths are hard to distinguish visually. For this reason, instead of plotting directly the persistence diagrams, we build a density estimation function for the set $M_D = \{d : (b, d) \in D \text{ and } d < +\infty\}$ and then we plot this function.

Definition 4.1. Let $D \subseteq \mathbb{R}$ be any finite set. The kernel density estimation function of the set D and the kernel K is the function

$$\rho_K(y, h) = \sum_{d \in M_d} K(y - d, h),$$

where K is a positive-defined kernel for $\mathbb{R} \times \mathbb{R}$, i.e., K is a symmetric function $K : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ such that $\sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j K(x_i, x_j) \geq 0$ for any $x_1, \dots, x_n \in \mathbb{R}$ given $n \in \mathbb{N}$ and $c_1, \dots, c_n \in \mathbb{R}$.

In our case, we use the kernel $K(x, h) = e^{-\frac{x^2}{2h^2}}$ with $h = h' = n^{-\frac{1}{5}}$. In particular, we denote by DensityFunction(D) any plot of the kernel density estimation function for a set D estimated with the previous K and $h = h'$ where the x-axis is used for the density function and the y-axis is used for the values of D. It can be proved that the function $\rho_K(y, h)|_{h=h'}$ is actually a density function in the probabilistic sense. See [38] an implementation and documentation of `seaborn.kdeplot` for more details.

4.4 Experiments

The dataset $\mathcal{D}$ we use in our experiments is a reduced version of the classical MNIST [25], that contains 28×28 pixels grayscale images. Our dataset is split into two disjoint datasets, the training dataset $\mathcal{D}_{\text{train}}$, containing $24,000$ images and the test dataset $\mathcal{D}_{\text{test}}$, containing $6,000$ images. To do that, we simply select the first $24,000$ and $6,000$ images from the original training dataset with $60,000$ images, as the original dataset is already sampled. We also split the dataset $\mathcal{D}_{\text{train}}$ into 48 disjoint subdatasets $\mathcal{D}_{\text{it}_1}, \dots, \mathcal{D}_{\text{it}_{48}}$ containing 500 examples each.

We build 10 different multilayer perceptrons $\text{NN} = \{\mathcal{N}_1, \dots, \mathcal{N}_{10}\}$ for the experiments using the Algorithm 3 with the inputs $n = 10$, $h_{\text{min}} = 50$, $h_{\text{max}} = 500$, $M = 1000$, $o = 10$ and $\sigma = \text{ReLU} \colon x \in \mathbb{R} \mapsto \max(0, x) \in \mathbb{R}$. Note that the ReLU function is $\mathcal{C}^r$ everywhere except on zero. Although it does not fit directly in our definition of neural network, it is the most used non-linear activation in deep learning and it usually outperforms other well-suited activation functions. The only problem of not having $\mathcal{C}^r$ functions f_θ^j in neural networks is that the gradient cannot be computed in certain points using the backpropagation algorithm 2. Most deep learning frameworks set $\text{ReLU}'(0) = 0$ [3] and then compute gradients by using the usual backpropagation scheme. In general, doing this does not affect to the training process, as the derivative of ReLU in zero is hardly ever used and also the numerical errors introduced by this substitution are not so big to affect the training process.

We generate 14 different experiments, summarised in Table 1. Each experiment trains the 10 neural networks generated previously during 15 epochs. Each epoch executes 48 training iterations, each trained with a different dataset $\mathcal{D}_{\text{it}_i}$, $i \in \{1, \dots, 48\}$, to compute the loss and the whole dataset $\mathcal{D}_{\text{train}}$ to compute the regularisation term. Also, we save all the accuracies computed in the datasets $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$ in each iteration. Finally, each five iterations, starting from the initial neural network $\mathcal{N}$, we save the zero dimension persistence diagram density functions of the network functional graph computed with the whole dataset $\mathcal{D}_{\text{train}}$. The training procedure for a neural network $\mathcal{N}_i$, $i \in \{1, \dots, 10\}$ and an experiment i is overviewed in Algorithm 4. Note that in our training procedure we use Adam as an updater of the neural network. The function Adam receives a neural network $\mathcal{N}$, a function $\mathcal{L}'$ whose inputs are the parameters of the neural network $\mathcal{N}$, and a dropout percentage R. The function applies the Adam algorithm implemented in the signature `tf.keras.optimizers.Adam` of the package Tensorflow [27], introduced for the first time in [24]. Note that if the dropout percentage R is greater than zero, the dropout method explained in 4.2 is applied to the neural network to compute the value $\mathcal{L}_{|_{D=\mathcal{D}_{\text{it}_i}}}(\theta)$ in $\mathcal{L}'(\theta)$, not affecting the part of the regularisation term during the Adam algorithm.

4.5 Results

Tables 2 contain the best validation and train accuracies obtained in each experiment during the training procedures together with the average accuracy per experiment. We can observe that the average training and validation accuracies for all the experiments performed with the standard deviation regularisation term (57) (experiments 3 to 8) are higher than the average accuracies of the experiments 1 and 2, that were trained only with the classical loss and either without dropout or with dropout generalisation respectively. This happens in general for every particular network: experiments trained with the standard deviation regularisation term consistently improve the results yielded by experiments trained only with a classical loss (with and without dropout), where only few examples behave differently. In particular, experiment 8, trained with a 10% of dropout and 30% of topological regulariser in the final loss was the experiment with best accuracies both in training and validation

Algorithm 3: Construction of neural networks

input : A number $n \in \mathbb{N}$ representing the number of neural networks to generate. Numbers $h_{\min}, h_{\max} \in \mathbb{N}$ representing the minimum and maximum number of vertices per hidden layer (both included). A number M representing the maximum number of neurons. A number $i \in \mathbb{N}$ representing the number of input vertices in the computational graph. A number $o \in \mathbb{N}$ representing the number of output vertices in the computational graph. A non-linear $\mathcal{C}^r$ function $\sigma : \mathbb{R} \to \mathbb{R}$.

output: A set $\mathrm{NN} = \{\mathcal{N}_1, \dots, \mathcal{N}_n\}$ of fully connected multilayer perceptrons.

$\mathrm{NN} \leftarrow \{\}$
for $j \leftarrow 1$ **to** n **do**
  $h \leftarrow 0$, $S \leftarrow \{\}$, $r \leftarrow M - o$
  while $r > 0$ *and* $r \geq h_{\min}$ **do**
    $h \leftarrow h + 1$
    $S \leftarrow S \cup \{\min(\mathrm{r}, \mathrm{randint}\,(\mathrm{h}_{\min}, \min\,(\mathrm{r}, \mathrm{h}_{\max})))\}$ `/* randint(a, b) samples uniformly a random integer in [a, b]. */`
  end
  for $w \leftarrow i + 1$ **to** $i + h_n + o$ **do**
    $\theta_w \leftarrow \mathrm{rand}\,()$ `/* rand() generates a random vector` $\theta_w \in \mathbb{R}^{\bar{d^w}+1}$`. It depends on the implementation of Tensorflow [27]. */`
    $f_\theta^w \leftarrow \sigma\left(\sum_{k=1}^{\bar{d^w}} \theta_{w,k} x_k + \theta_{w,\bar{d^w}+1}\right)$
  end
  $\mathcal{N}_j \leftarrow \left(i, h, S, o, E, (0)_{i \in \{1,\dots,i+h_n+o\}}, \left(f_\theta^j\right)_{j \in \{1,\dots,i+h_n+o\}}, (\theta_j)_{j \in \{1,\dots,i+h_n+o\}}\right)$.
  $\mathrm{NN} \leftarrow \mathrm{NN} \cup \{\mathcal{N}_j\}$
end
return NN

Algorithm 4: Training procedure for each neural network in each experiment

input : A neural network $\mathcal{N}$ and a number of experiment i.

output: An updated neural network $\mathcal{N}$ with *optimal* parameters and, two lists of accuracies $\text{Acc}_{\text{train}}$, Acc_{test} per training iteration and a set `PPDD` of zero dimension persistence diagrams density functions.

$\text{Acc}_{\text{train}}, \text{Acc}_{\text{test}}, \text{PPDD} \leftarrow \{\}, \{\}, \{\}$

$j \leftarrow 0$ `/* Counts the number of iterations. */`

$R \leftarrow r$ where r is the dropout percentage of the experiment i in 1.

for $e \leftarrow 1$ **to** 15 **do**

  for $i \leftarrow 1$ **to** 48 **do**

    $\mathcal{L}' \leftarrow (1 - (w/10))\mathcal{L}_{|_{D=\mathcal{D}_{\text{it}_i}}} + (w/10)\mathcal{T}_{|_{D=\mathcal{D}_{\text{train}}}}$ the final loss function of the experiment i as explained in 1.

    if $j = 0$ *mod* 5 **then**

      $\text{PPDD} \leftarrow \text{PPDD} \cup \{\text{DensityFunction}(\{d : (b, d) \in \text{Dgm}(\mathcal{N}, \mathcal{D}_{\text{train}})\})\}$ where $\text{Dgm}(\mathcal{N}, \mathcal{D}_{\text{train}})$ is as in 4.1.

    end

    $\mathcal{N} \leftarrow \text{Adam}(\mathcal{N}, \mathcal{L}', R)$

    $\text{Acc}_{\text{train}} \leftarrow \text{Acc}_{\text{train}} \cup \{\text{Accuracy}(\mathcal{N}, \mathcal{D}_{\text{train}})\}$

    $\text{Acc}_{\text{test}} \leftarrow \text{Acc}_{\text{test}} \cup \{\text{Accuracy}(\mathcal{N}, \mathcal{D}_{\text{test}})\}$

    $j \leftarrow j + 1$

  end

end

return $(\mathcal{N}, \text{Acc}_{\text{train}}, \text{Acc}_{\text{test}}, \text{PPDD})$

Experiment	Dropout (%)	Topological regulariser (%)	Topological regulariser $\mathcal{T}$
1	0	0	-
2	10	0	-
3	0	10	$\mathcal{T}_{\text{std}}$
4	10	10	$\mathcal{T}_{\text{std}}$
5	0	20	$\mathcal{T}_{\text{std}}$
6	10	20	$\mathcal{T}_{\text{std}}$
7	0	30	$\mathcal{T}_{\text{std}}$
8	10	30	$\mathcal{T}_{\text{std}}$
9	0	10	$\mathcal{T}_{\text{red}}$
10	10	10	$\mathcal{T}_{\text{red}}$
11	0	20	$\mathcal{T}_{\text{red}}$
12	10	20	$\mathcal{T}_{\text{red}}$
13	0	30	$\mathcal{T}_{\text{red}}$
14	10	30	$\mathcal{T}_{\text{red}}$

Table 1: Table summarising the most important differences between experiments. The *Dropout (%)* column indicates the chances (in percentage) we assign to each vertex to be ignored during an iteration of the training procedure, as explained in 4.2. The *Topological regulariser (%)* column indicates how much the topological regulariser affects the loss used in the experiment. The column *Topological regulariser* $\mathcal{T}$ indicates the topological regulariser used in the experiment. The value "-" in this column indicates that no topological regulariser is used in the experiment. More details of the specific training procedure for each experiment can be found in 4.

datasets. This means that both regularisers are not exclusive and when used together can improve further the results yielded by either of them individually. However, note that in both, validation and training, experiments trained with the standard deviation regularisers and without dropout performs almost as well as experiment 8, as experiments 5 and 7, well ahead of the accuracies of experiments 0 and 1, making this topological regulariser useful individually.

The sampled topological redundancy generalisation term 58 worked in a detrimental way, yielding accuracies approaching to 0.1 (10%), the expected accuracy for random guesses in classification problems with 10 classes. This could be interpreted by observing Plots 1, where density functions of dimension zero persistence diagrams are plotted for each experiment in the iterations with best and worse accuracies in validation. Here, we can observe that well-trained networks have two different clusters regarding neuron correlation, one where activation vectors are highly correlated and another one where activation vectors are loosely correlated. In the graphic, this seems to be an invariant of a well trained network and it is not true for the plots of the worse validation accuracies. Minimising topological redundancy implies removing one of these clusters (the one with high correlations in absolute value) and this may be causing the detriment of neural network accuracies when training with this regularisation term.

5. Conclusions

We proved that classical stability and differentiability results can be extended to more general cases of symmetric functions when building persistence diagrams of Vietoris-Rips sublevel set filtrations. This is important because in real scenarios similarities of interest between objects may not be captured by real distance functions. This is the case of the symmetric function $d'(x, y) = 1 - |\text{Corr}(x, y)|$, used in [1] to build persistence diagrams that, under some transformations, are able to assess the generalisation capacity of neural networks.

As suggested in [1], we saw that standard deviations of persistence diagrams are directly related to the generalisation capacity of neural networks. In particular, we saw that minimising the standard deviation of deaths of zero dimension persistence diagrams together with a usual loss function consistently yielded better results than minimising only the loss function. In some examples, this regulariser outperformed the dropout method, considered one of the fundamentals regularisers when training neural networks.

Topological redundancy did not work as expected. Although it is based on a theoretical result, it is probable that error propagation affects more than supposed initially. Also, Plots 1 show us that trained neural networks tend to generate two opposed clusters in the density function of zero dimension persistence diagrams of network functional graphs, one with high values of deaths and another with low ones. Minimising the topological redundancy results in the elimination of one of these clusters and thus the modification of one of the observed *invariants* of well-trained neural networks.

Regarding these two clusters in the persistence diagrams of network functional graphs, they were consistently generated in all the neural network training processes we executed. This is probably not a coincidence and further study of its meaning is of interest to the deep learning research community to develop stronger generalisation measures.

In general, we showed that topological data analysis is useful to build a framework for neural network *interpretability* that can be integrated into the training processes to improve neural network generalisation capacities. To the best of our knowledge, this is the first work in which the structure of activation vectors of neural networks is used to add a generalisation measure as a loss term in the literature.

5.1 Future work

Several aspects of this master thesis can be improved and further studied. As an example, the stability results for symmetric functions coming from a distance presented in Subsection 2.2.1 are limited to functions d' that are Hölder transformations of true distances d. However, we showed in Subsection 2.2.2 that differentiability is not limited to this kind of transformations and we conjecture that more general results depending only on the continuity of functions γ can be given.

Automatic differentiation (AD) is a key of the computational part of the thesis. AD is used to compute the gradients in each step of the training process and its use is the usual approach to compute gradients of loss functions coming from persistence diagrams. However, automatic differentiation can introduce problems that are difficult to debug. An specific persistence diagram differentiation codebase that takes advantage of the closed formulas for the differentials of r-differentiable functions $\mathcal{M} \to \mathbf{Bar}_\Delta$ and $\mathbf{Bar}_\Delta \to \mathcal{N}$ would be especially useful for the topological data analysis community.

We proposed topological redundancy as a topological regularisation term based on the redun-

dancy of neurons having correlation in absolute value equal to one. However, we did not compute the error in the predictions introduced when removing neurons that have correlation in absolute value different to one. Knowing these values would help us to design better topological redundancy regularisation terms based on these errors.

In the thesis we analysed only two regularisation terms based on very specific settings of Vietoris-Rips sublevel set filtrations. However, there are many *distances* (symmetric functions) that we can use to build our Vietoris-Rips filtrations. It seems that Information Geometry could be key in this case, as it is being used satisfactorily to build manifolds of neural networks with *metrics* given by some symmetric functions called divergences, that are in the core of this discipline. With new approaches we could get nearer to the fundamental geometrical properties that define robust neural networks and build regularisation terms accordingly.

Finally, there is a lot of work optimising our methods to work on a larger class of neural networks. Currently, the thesis is focused on multilayer perceptrons, although the notions introduced in the thesis can be extended to the majority of neural networks. The problem is given by the number of vertices the algorithm can handle for a specific neural network. Currently, networks with more than 5000 vertices cannot be handled properly by the methods presented here due to the high complexity of the algorithm that computes persistence diagrams, that is $\mathcal{O}(n^3)$ in time and memory where n is the number of vertices of the neural network. A good approach would be to use the sampling strategies used in [1]. Also, being able to work with GPUs, as in Ripser++ [39], would improve the speed of our algorithms, and therefore its usefulness.

In general much work remains to be done. However, we have developed a theoretical framework that could allow deep learning practitioners to analyse further their neural network implementations, with the hope that more precise and robust generalisation metrics will be developed in the future inspired from this work.

Bibliography

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

[2] Ulrich Bauer. "Ripser: efficient computation of Vietoris–Rips persistence barcodes". In: *Journal of Applied and Computational Topology* 5.3 (Sept. 2021), pp. 391–423. ISSN: 2367-1734. DOI: 10.1007/s41468-021-00071-5. URL: https://doi.org/10.1007/s41468-021-00071-5.

[3] David Bertoin et al. "Numerical influence of ReLU'(0) on backpropagation". In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato et al. Vol. 34. Curran Associates, Inc., 2021, pp. 468–479.

[4] D. Burago et al. *A Course in Metric Geometry*. Crm Proceedings & Lecture Notes. American Mathematical Society, 2001. ISBN: 9780821821299.

[5] Gunnar Carlsson et al. "Persistence Barcodes for Shapes". In: *International Journal of Shape Modeling* 11.02 (2005), pp. 149–187. DOI: 10.1142/S0218654305000761. eprint: https://doi.org/10.1142/S0218654305000761. URL: https://doi.org/10.1142/S0218654305000761.

[6] Mathieu Carriere et al. “Optimizing persistent homology based functions”. In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, 18–24 Jul 2021, pp. 1294–1303. URL: https://proceedings.mlr.press/v139/carriere21a.html.

[7] Jeremy Charlier, Radu State, et al. “PHom-GeM: Persistent Homology for Generative Models”. In: *The 6th Swiss Conference on Data Science (SDS), 2019 IEEE International Conference*. IEEE. 2019.

[8] Frédéric Chazal, Vin De Silva, and Steve Oudot. “Persistence stability for geometric complexes”. In: *Geometriae Dedicata* 173.1 (2014), pp. 193–214.

[9] Frédéric Chazal et al. *The structure and stability of persistence modules*. Springer, 2016.

[10] Chao Chen et al. “A Topological Regularizer for Classifiers via Persistent Homology”. In: *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*. Ed. by Kamalika Chaudhuri and Masashi Sugiyama. Vol. 89. Proceedings of Machine Learning Research. PMLR, 16–18 Apr 2019, pp. 2573–2582. URL: https://proceedings.mlr.press/v89/chen19g.html.

[11] Michael Collins. *Notes on computational graphs, and backpropagation*. http://www.cs.columbia.edu/~mcollins/cs4705-spring2020/. COMS W4705: Natural Language Processing. 2020.

[12] William Crawley-Boevey. “Decomposition of pointwise finite-dimensional persistence modules”. In: *Journal of Algebra and Its Applications* 14.05 (2015), p. 1550066.

[13] Herbert Edelsbrunner and John Harer. *Computational Topology - an Introduction.* American Mathematical Society, 2010, pp. I–XII, 1–241. ISBN: 978-0-8218-4925-5.

[14] Patrizio Frosini. “Measuring shapes by size functions”. In: *Intelligent Robots and Computer Vision X: Algorithms and Techniques*. Ed. by David P. Casasent. Vol. 1607. Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series. Feb. 1992, pp. 122–133. DOI: 10.1117/12.57059.

[15] Rickard Brüel Gabrielsson et al. “A Topology Layer for Machine Learning”. In: *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*. Ed. by Silvia Chiappa and Roberto Calandra. Vol. 108. Proceedings of Machine Learning Research. PMLR, 26–28 Aug 2020, pp. 1553–1563. URL: https://proceedings.mlr.press/v108/gabrielsson20a.html.

[16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[17] G. Henselman and R. Ghrist. “Matroid Filtrations and Computational Persistent Homology”. In: *ArXiv e-prints* (June 2016). arXiv: 1606.00199 [math.AT].

[18] Christoph Hofer et al. “Connectivity-Optimized Representation Learning via Persistent Homology”. In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, Sept. 2019, pp. 2751–2760. URL: https://proceedings.mlr.press/v97/hofer19a.html.

[19] Xiaoling Hu et al. “Topology-Aware Segmentation Using Discrete Morse Theory”. In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=LGgdb4TS4Z.

[20] Xiaoling Hu et al. “Topology-Preserving Deep Image Segmentation”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper/2019/file/2d95666e2649fcfc6e3af75e09f5adb9-Paper.pdf.

[21] Yiding Jiang et al. “Methods and Analysis of The First Competition in Predicting Generalization of Deep Learning”. In: *Proceedings of the NeurIPS 2020 Competition and Demonstration Track*. Ed. by Hugo Jair Escalante and Katja Hofmann. Vol. 133. Proceedings of Machine Learning Research. PMLR, June 2021, pp. 170–190. URL: https://proceedings.mlr.press/v133/jiang21a.html.

[22] Yiding Jiang et al. *NeurIPS 2020 Competition: Predicting Generalization in Deep Learning*. 2020. DOI: 10.48550/ARXIV.2012.07976. URL: https://arxiv.org/abs/2012.07976.

[23] Yiding Jiang* et al. “Fantastic Generalization Measures and Where to Find Them”. In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=SJgIPJBFvH.

[24] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015. URL: http://arxiv.org/abs/1412.6980.

[25] Yann LeCun and Corinna Cortes. “MNIST handwritten digit database”. In: (2010). URL: http://yann.lecun.com/exdb/mnist/.

[26] Jacob Leygonie, Steve Oudot, and Ulrike Tillmann. “A Framework for Differential Calculus on Persistence Barcodes”. In: *Foundations of Computational Mathematics* (July 2021). ISSN: 1615-3383. DOI: 10.1007/s10208-021-09522-y. URL: https://doi.org/10.1007/s10208-021-09522-y.

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

[28] Facundo Memoli. “Gromov-Hausdorff distances in Euclidean spaces”. In: *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*. 2008, pp. 1–8. DOI: 10.1109/CVPRW.2008.4563074.

[29] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[30] Jose A. Perea. *A Brief History of Persistence*. 2018. DOI: 10.48550/ARXIV.1809.03624. URL: https://arxiv.org/abs/1809.03624.

[31] Allan Pinkus. “Approximation theory of the MLP model in neural networks”. In: *Acta Numerica* 8 (1999), pp. 143–195. DOI: 10.1017/S0962492900002919.

[32] Vanessa Robins. “Towards computing homology from approximations”. In: *Topology Proceedings* 24 (Jan. 1999).

[33] “Simple Linear Regression”. In: *Linear Models in Statistics*. John Wiley & Sons, Ltd, 2007. Chap. 6, pp. 127–136. ISBN: 9780470192610. DOI: https://doi.org/10.1002/9780470192610.ch6. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9780470192610.ch6. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470192610.ch6.

[34] Victor Solo. *Pearson Distance is not a Distance*. 2019. arXiv: 1908.06029 [stat.ME].

[35] Nitish Srivastava et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[36] The GUDHI Project. *GUDHI User and Reference Manual*. GUDHI Editorial Board, 2015. URL: http://gudhi.gforge.inria.fr/doc/latest/.

[37] *Variance of LC of RVs*. https://www.projectrhea.org/rhea/index.php/Variance_of_LC_of_RVs. Accessed: 2022-08-22.

[38] Michael L. Waskom. “seaborn: statistical data visualization”. In: *Journal of Open Source Software* 6.60 (2021), p. 3021. DOI: 10.21105/joss.03021. URL: https://doi.org/10.21105/joss.03021.

[39] Simon Zhang, Mengbai Xiao, and Hao Wang. “GPU-Accelerated Computation of Vietoris-Rips Persistence Barcodes”. In: *36th International Symposium on Computational Geometry (SoCG 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik. 2020.

A. Full results of experiments

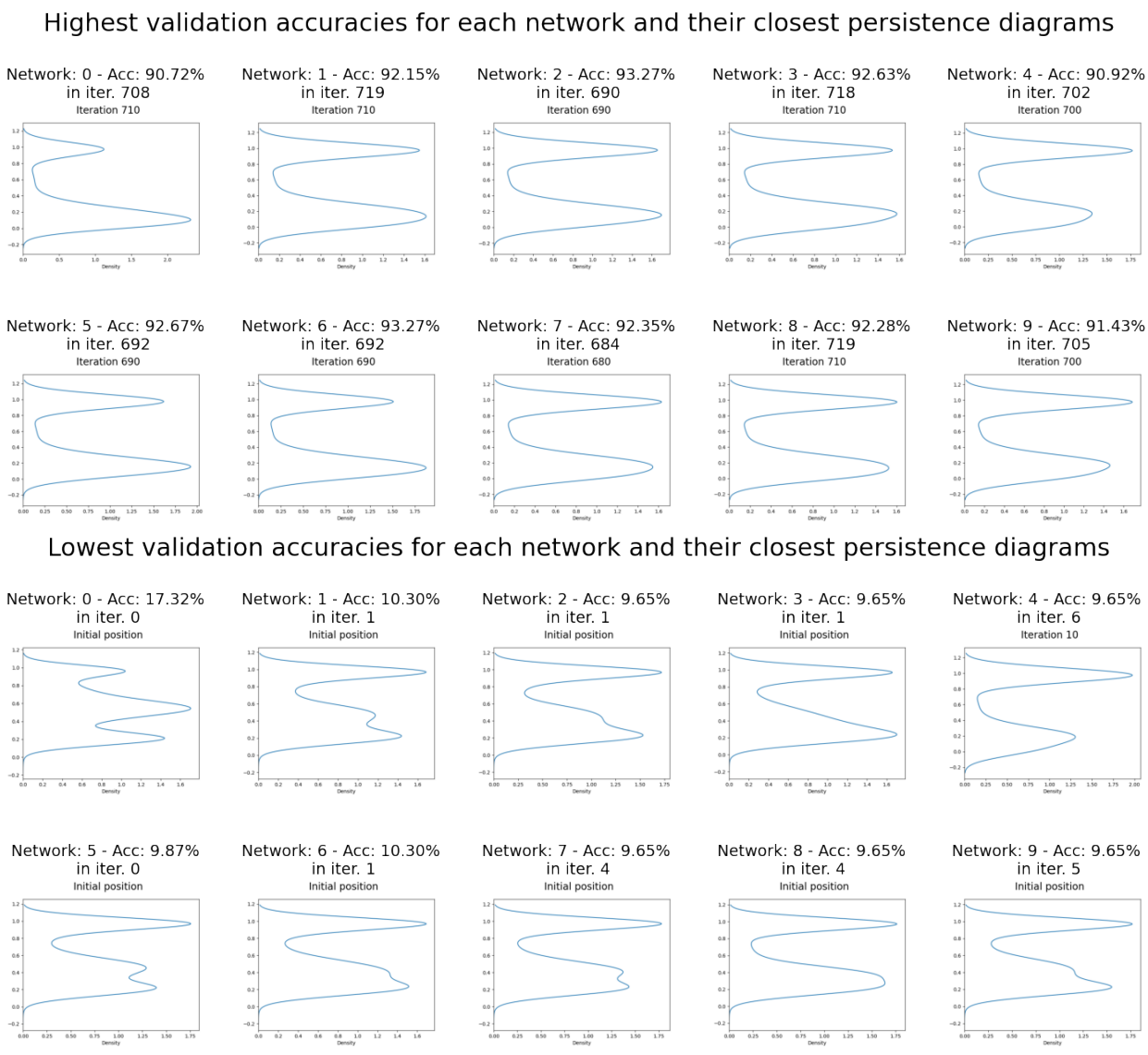


Figure 1: Density functions 4.3 of zero dimension persistence diagrams of network activation graphs of the 10 generated neural networks during the experiment 0 training in the closest iteration to the iteration where they obtained its highest and lowest validation accuracies, respectively.

Validation accuracies for networks in each experiment														
Net.	Ex. 1	Ex. 2	Ex. 3	Ex. 4	Ex. 5	Ex. 6	Ex. 7	Ex. 8	Ex. 9	Ex. 10	Ex. 11	Ex. 12	Ex. 13	Ex. 14
0	0.907	0.924	0.926	0.934	0.928	0.937	0.931	0.941	0.115	0.117	0.115	0.117	0.116	0.115
1	0.922	0.94	0.958	0.962	0.941	0.96	0.956	0.965	0.115	0.115	0.118	0.115	0.115	0.115
2	0.933	0.948	0.956	0.959	0.958	0.96	0.957	0.962	0.115	0.115	0.115	0.115	0.115	0.115
3	0.926	0.95	0.952	0.96	0.955	0.958	0.957	0.96	0.115	0.115	0.115	0.115	0.115	0.115
4	0.909	0.902	0.944	0.944	0.946	0.943	0.944	0.943	0.115	0.115	0.115	0.115	0.115	0.115
5	0.927	0.943	0.946	0.956	0.949	0.941	0.946	0.958	0.115	0.144	0.116	0.15	0.115	0.115
6	0.933	0.949	0.955	0.959	0.958	0.959	0.956	0.961	0.115	0.136	0.115	0.115	0.115	0.115
7	0.924	0.939	0.957	0.961	0.956	0.959	0.956	0.958	0.115	0.115	0.115	0.115	0.115	0.115
8	0.923	0.93	0.95	0.953	0.949	0.954	0.95	0.952	0.115	0.115	0.115	0.115	0.115	0.115
9	0.914	0.944	0.94	0.959	0.956	0.959	0.957	0.955	0.115	0.117	0.115	0.115	0.115	0.115
Avg.	0.922	0.937	0.949	0.955	0.9500	0.953	0.951	0.955	0.115	0.120	0.116	0.119	0.115	0.155

Training accuracies for networks in each experiment														
Net	Ex. 1	Ex. 2	Ex. 3	Ex. 4	Ex. 5	Ex. 6	Ex. 7	Ex. 8	Ex. 9	Ex. 10	Ex. 11	Ex. 12	Ex. 13	Ex. 14
0	0.922	0.935	0.938	0.947	0.943	0.95	0.945	0.953	0.114	0.115	0.114	0.115	0.114	0.114
1	0.941	0.958	0.979	0.979	0.959	0.978	0.977	0.98	0.114	0.114	0.114	0.114	0.114	0.114
2	0.952	0.968	0.976	0.979	0.977	0.98	0.979	0.981	0.114	0.114	0.114	0.114	0.114	0.114
3	0.95	0.968	0.974	0.978	0.974	0.976	0.976	0.977	0.114	0.114	0.114	0.114	0.114	0.115
4	0.928	0.92	0.967	0.96	0.965	0.962	0.961	0.959	0.114	0.114	0.114	0.114	0.114	0.114
5	0.94	0.957	0.965	0.972	0.968	0.951	0.963	0.973	0.114	0.137	0.114	0.138	0.114	0.114
6	0.947	0.963	0.976	0.977	0.979	0.974	0.976	0.979	0.114	0.119	0.114	0.114	0.114	0.115
7	0.944	0.96	0.981	0.977	0.975	0.978	0.975	0.977	0.114	0.114	0.114	0.114	0.114	0.114
8	0.944	0.945	0.973	0.969	0.968	0.969	0.969	0.967	0.114	0.114	0.114	0.114	0.114	0.114
9	0.943	0.962	0.956	0.977	0.978	0.978	0.98	0.971	0.114	0.115	0.114	0.114	0.114	0.114
Avg.	0.941	0.954	0.969	0.971	0.970	0.970	0.970	0.972	0.114	0.117	0.114	0.117	0.114	0.115

Table 2: Maximum validation and training accuracies achieved during the training of the 10 generated neural networks, listed from 0 to 9, for all the 14 experiments performed, described in 1. To obtain these, we computed both types of accuracies of the given network in the given experiment for each iteration of the training procedure. Experiments 1 and 2 were trained only with the classical loss either without or with regularisation term, respectively. Experiments from 3 to 8 were trained with the classical loss and different combinations of the standard deviation term (57) percentage and of the dropout percentage. Experiments from 9 to 14 were trained with different combinations of the topological redundancy term (58) percentage and of the dropout percentage. Notably, experiments trained with the standard deviation term consistently outperform the other experiments with respect to their accuracies, both individually for each network and globally comparing the average accuracies per experiment. Experiments trained with the topological redundancy dropped their performance dramatically showing us that having neuron activation redundancies is important when networks generalise and further study is required to understand how these redundancies affect neural networks.