UNIVERSITAT DE BARCELONA

Treball final de grau

DOBLE GRAU DE MATEMÀTIQUES I ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

Topological Descriptors in Deep Learning and Their Connection with Model Generalization

Autor: Rubén Ballester Bautista

Contents

Abstract **iii**

1 Introduction **1**

2 Background theory **7**
2.1 Deep learning . . . 7
2.1.1 Computational graph . . . 7
2.1.2 Deep learning basics . . . 9
2.1.3 Multilayer perceptrons . . . 10
2.2 Basic homology . . . 13
2.2.1 Simplices . . . 13
2.2.2 Simplicial complexes . . . 15
2.2.3 Abstract simplicial complexes . . . 15
2.2.4 Filtrations . . . 15
2.2.5 Homology groups . . . 17
2.3 Persistent homology . . . 19
2.3.1 Comparison of persistence diagrams . . . 22
2.3.2 Topological descriptors . . . 23

3 Analysis, design and implementation **31**
3.1 Development of a dataset . . . 32
3.1.1 IMDB reviews iterations choice . . . 34
3.1.2 CIFAR10 iterations choice . . . 35
3.2 Computation of persistence diagrams . . . 38
3.3 Predicting generalization gap using topological features . . . 39

4 Results **43**
4.1 Generalization gap distributions . . . 43
4.2 Visualizations and final results . . . 44
4.2.1 Leading coefficients of complex polynomials . . . 45
4.2.2 Context dependence in generalization analysis . . . 46

5 Conclusions **47**
5.1 Future work . . . 48

Bibliography **51**

Appendices **55**
A.1 Objectives and planification 55
A.2 Figures from results 55

Abstract

Deep neural networks (DNNs) have become the dominant machine learning paradigm in the past decade due to their increased ability of learning representations of high complexity models to make predictions or infer data. Due to their ease of use and versatility, they have become the preferred solution for many different statistical learning problems like Object Recognition from images and Sentiment Analysis from text. Unfortunately, the increase in performance has come with a cost. DNNs nowadays are way too complicated for humans to understand as they optimize up to millions of parameters that define a very complex computational graph, through which a high number of intermediate representations are built at inference time. This rises a somehow paradoxical situation. Even though we define the initial state of the model and control the optimization of its parameters (what Machine Learning practitioners generally call *training*), at the end of this process we are left with what is generally regarded as a black box, a computational graph whose internal representations we cannot interpret. The only option to analyze DNN performances at this point is to measure the overall prediction performance without having access to the DNN inner representation that actually gave rise to that performance. Understanding the internal mechanisms of DNNs and how they relate with this performance is of fundamental importance for several reasons, such as the possibility of selecting the best network structure according to its generalization capacity in order to increase the model's generalization in semantically similar but previously unseen data, the design of more computationally efficient models adapted to specific problems, the automation of deep learning model selection, etc.

In this work we study the quality of deep representations using Algebraic Topology and, in particular, Topological Data Analysis. We are interested in the ability of a deep neural network representation to generalize to unseen samples, as measured by the generalization gap, the absolute difference in performance between the data available in training and a separate data partition only available after the training phase has concluded, usually called test set.

Topological Data Analysis (TDA) studies topological properties of finite sets of data. Therefore, given an appropriate representation of a computational graph, TDA techniques, such as persistent homology, analyze rigorously the *shape* of this representation, extracting useful features that can be ultimately compared and, more importantly, interpreted in general. In our particular case, we will work with weighted nonnegative fully connected graphs, arising from pseudo-semimetrics, i.e., metrics which do not necessarily respect the triangle inequality and may fail to be positive definite.

We study how well topological descriptors of a given DNN predict the generalization gap on two machine learning problems: Object Recognition from images and Sentiment Analysis from text. We show that, although we have not found a unified model that describes generalization gaps using persistent homology, TDA techniques yield promising results predicting generalization gaps, with high correlations between generalization gap distributions and several linear models trained with topological features.

Resumen

Las redes neuronales profundas (RNP) se han convertido en el paradigma dominante en aprendizaje automático en la pasada década debido a su gran capacidad para aprender representaciones de modelos complejos para inferir datos. Debido a su versatilidad, se han convertido en la solución preferida para diversos problemas estadísticos como el reconocimiento de objetos en imágenes o el análisis de sentimientos. Desafortunadamente, las mejoras descritas por este paradigma han venido acompañadas de ciertos desafíos. Actualmente, las representaciones realizadas por las RNPs son demasiado difíciles de interpretar para los humanos, con millones de parámetros que definen estructuras computacionales en las que cientos de representaciones internas son aprendidas sin supervisión ni razonamiento humano. De alguna manera, esto define una situación paradójica, ya que, aunque podamos controlar ciertos aspectos como el estado inicial de las redes o ciertos parámetros (lo que generalmente es conocido como *entrenamiento* en el aprendizaje automático), el resultado final del proceso es un modelo predictivo cuyas representaciones internas no podemos comprender (esto se conoce como el problema de la caja negra del aprendizaje profundo). De hecho, una vez realizado el entrenamiento, la única manera posible de comprobar la capacidad predictiva del modelo es utilizar métricas sobre las predicciones, y no sobre el modelo en sí. Entender los mecanismos internos de las RNPs y cómo se relacionan con esta capacidad predictiva es de una importancia fundamental por diferentes razones, ya que el análisis interno del modelo podría permitir a los usuarios seleccionar el mejor modelo a partir de la capacidad de generalización para elementos no vistos previamente por el modelo para un problema dado de un conjunto de modelos arbitrarios, permitir diseños específicos para modelos concretos, permitir la automatización de *pipelines* de aprendizaje profundo, etc.

En este trabajo estudiaremos la calidad de las representaciones de diferentes redes neuronales usando Análisis de Datos Topológico. Estaremos interesados en la capacidad de las representaciones para ser generalizadas a nuevos ejemplos, medida mediante una métrica llamada *brecha de generalización*, consistente en la diferencia absoluta de rendimiento en el conjunto de entrenamiento y un conjunto de datos no utilizados en la fase de entrenamiento, llamado conjunto de prueba.

El Análisis de Datos Topológico (ADT) estudia propiedades topológicas de conjuntos finitos de datos. Dada una representación apropiada de un grafo computacional, las técnicas de ADT, como la homología persistente, serán usadas para analizar rigurosamente la *forma* de esta representación, extrayendo de ella características útiles que puedan ser comparadas, y, de forma más importante, interpretadas, en general. En nuestro caso particular, trabajaremos con grafos no negativamente ponderados finitos, una construcción basada en pseudosemimétricas, una relajación de la definición usual de métrica.

Estudiaremos con qué precisión los descriptores topológicos predicen la brecha de generalización en dos problemas de aprendizaje automático: reconocimiento de objetos a partir de imágenes y análisis de sentimientos a partir de texto. Además, mostraremos que, pese a que no hemos encontrado un modelo unificado que describa las brechas de generalización usando homología persistente, las técnicas de ADT generan resultados prometedores para predecir las brechas de generalización, con altas correlaciones entre ellas y diferentes modelos lineales entrenados con características topológicas.

Resum

Les xarxes neuronals profundes (XNP) s'han convertit en el paradigma dominant en aprenentatge automàtic en la dècada passada a causa de la seva capacitat per aprendre representacions de models complexos per realitzar prediccions o inferir-ne dades. Gràcies a la seva facilitat d'ús i versatilitat, s'han convertit en la solució predilecta per a diversos problemes estadístics, com el reconeixement d'objectes a partir d'imatges o l'anàlisi de sentiments en textos. Desafortunadament, les millores descrites per aquest paradigma han estat acompanyades d'alguns reptes. Actualment, les representacions realitzades per les XNP són massa difícils d'interpretar pels éssers humans, amb milions de paràmetres que defineixen estructures computacionals complexes en què centenars de representacions internes són apreses sense supervisió ni raonament humà. D'alguna manera, això defineix una situació paradoxal, ja que, encara que podem controlar certs aspectes com l'estat inicial de les xarxes o certs paràmetres de l'optimització de la seva estructura (cosa que generalment és conegut com a *entrenament* en l'aprenentatge automàtic), el resultat final del procés és un model predictiu les representacions internes del qual no podem comprendre (aquest problema se sol anomenar el problema de la caixa negra de l'aprenentatge profund). De fet, un cop realitzats els entrenaments, l'única manera possible de comprovar la capacitat predictiva del model és utilitzar mètriques sobre les prediccions, i no sobre el model en ell mateix. Entendre els mecanismes interns de les XNPs i com es relacionen amb aquesta capacitat predictiva és d'una importància fonamental per diferents raons, ja que l'anàlisi interna del model podria permetre als usuaris seleccionar el millor model a partir de la capacitat de generalització per a elements no vistos prèviament pel model en un problema donat d'un conjunt de dades arbitrari, permetent dissenys específics per a models concrets, permetent l'automatització de *pipelines* d'aprenentatge profund, etc.

En aquest treball, estudiarem la qualitat de les representacions de diferents xarxes neuronals usant Anàlisi de Dades Topològica. Concretament, estarem interessats en la capacitat de les representacions per generalitzar-se a nous exemples, mesurada mitjançant una mètrica anomenada *bretxa de generalització*, consistent en la diferència absoluta de rendiment en el conjunt d'entrenament i un nou conjunt de dades no utilitzades prèviament.

L'Anàlisi de Dades Topològica (ADT) estudia propietats topològiques de conjunts finits de dades. Donada una representació adequada d'un graf computacional provinent d'un conjunt de neurones, les tècniques d'ADT, com ara l'homologia persistent, seran utilitzades per analitzar rigorosament la *forma* d'aquesta representació, extraient-ne característiques útils que puguin ser finalment comparades, i encara més important, interpretades, en general. En el nostre cas, treballarem amb grafs finits i no negativament ponderats, una construcció basada en pseudosemimètriques, una relaxació de la definició de mètrica.

Estudiarem amb quina precisió els descriptors topològics prediuen la bretxa de generalització en dos problemes d'aprenentatge automàtic: reconeixement d'objectes a partir d'imatges i anàlisi de sentiments a partir de text. A més a més, mostrarem que, encara que no hem trobat un model unificat que descrigui les bretxes de generalització utilitzant homologia persistent, les tècniques d'ADT generen resultats esperançadors per predir les bretxes de generalització, amb altes correlacions entre aquestes bretxes i diferents models lineals entrenats amb característiques topològiques.

2010 Mathematics Subject Classification. 55N35

Chapter 1

Introduction

Deep learning is a subfield of machine learning concerned with algorithms that initially tried to replicate the structure and functionalities of a human brain but has gone beyond these anatomical replications. To do that, deep learning uses neural networks, that are ultimately interconnected functions that learn the latent source of truth (distribution) of the data we want to learn about; see figure 1.1. Once networks have learnt a model, it is interesting to see if this model is really working properly in unseen data, that is, generalizing properly. According to [19]: "*Understanding generalization in neural networks is arguably one of the most important questions in deep learning*". But, what does **generalization** mean? We can intuitively define generalization as the capacity of a model to adapt properly to new data drawn from the same latent source of truth (distribution) used to create the model; see figure 1.2. Deep learning has achieved numerous successful applications in the latest years, as the recent application of deep learning to give a solution to a 50-year-old problem in Biology [20]. However, understanding why a model generalizes better than others is a complex challenge because of the high complexity of network structures (known as the deep learning black box problem). The higher the complexity of a network, the harder to interpret what specific parts of the network do. In this topic, several studies have been published, such as the one that uses a technique called **activation maximization**, that tries to understand what a neural network learns by finding the input that maximizes a class in a classification problem [30], or the ones published by Ciprian Corneanu [10, 11] that use some techniques from topological data analysis to predict the generalization gap, a measure that is related directly with the capacity of a network [19].

Precisely, topological data analysis (TDA) is a subfield of applied mathematics that applies topological techniques to the study of datasets, usually given as point clouds (finite sets of points) equipped with a metric (although in our case we will be using weighted nonnegative fully connected graphs). TDA has been growing rapidly during the last years due to its practical applications and nice properties, that allow a *fast* computation of topological descriptors of data that can be used in a number of techniques, like as input for machine learning models. In fact, in both papers from Corneanu, we can see a novel connection between a neural network problem (dataset and neural network structure) and TDA. This connection consists of the **activation graph**, that is a weighted nonnegative fully connected graph created from the neurons of the networks evaluated in the training

dataset points. There, he analyzed persistent homology of this space, connecting some properties of persistent homology (life and midlife) to the generalization gap metric using linear regression. Applications of algebraic topology (persistent homology) have been growing fastly during the last years, with works that analyze actual neuron correlations [16] in rats, temporal graphs [23] or optimization problems [32], among others. To do that, researchers try to get nice topological features from objects called persistence diagrams, that can be easily computed from a finite set of points that can be presented in many different structures, like metric spaces or weighted graphs; see 1.3.

In this project, I will develop the topic that Corneanu et al. started, defining concisely and precisely what the activation graph is, implementing a software capable of analyzing arbitrary neural networks and trying to improve the state of the art using more sophisticated techniques of regression and using features mapped from persistence diagrams, the main tool of TDA.

To do that, we will first generate two different datasets of neural networks. Each dataset will consist of a classical problem dataset (CIFAR10 and IMDB reviews sentiment analysis, split in train, validation and test datasets) and a set of synthetic neural networks generated automatically. For each problem, we will create 60 different neural network structures and then we will keep underfitted, nice fitted and overfitted versions of these 60 different structures (see chapter 3). Once these datasets are generated, we will compute the activation graph using a modification of the correlation measure. Then, we will compute persistence diagrams for some dimensions, in particular 0, 1 and 2 (as experimentally these are the ones that gave more information about neural networks). Finally, we will map these persistence diagrams to better representations to train intuitive linear models that approximate the generalization gap function, seeking to obtain a relation between persistent homology and generalization.

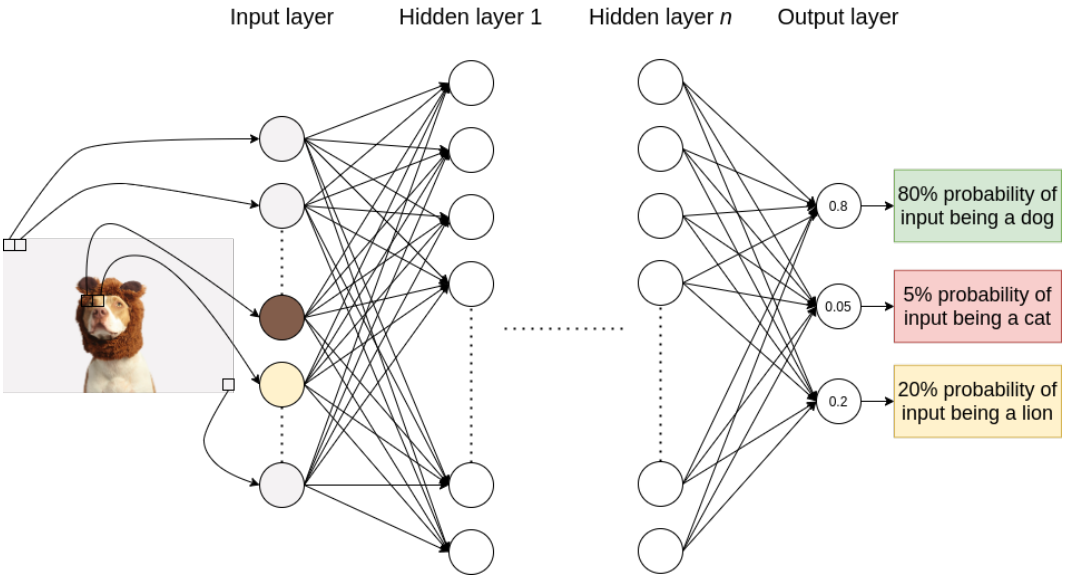


Figure 1.1: A fully connected neural network (multilayer perceptron, MLP) of n hidden layers, whose input is a vector in $\mathbb{R}^{n \cdot m}$ representing the pixel values of images of width n and height m, modelling a function that classifies images in three different categories, namely *cats*, *dogs* and *lions*. To classify images, the neural network outputs a vector in $\mathbb{R}^3$ where the first, second and third values are the probabilities of an image representing a dog, a cat and a lion respectively. A neural network, in general, is a directed graph where each vertex value, given an input, is computed as a function of the values of the vertices pointing to it with a directed edge, except the input vertices that are filled with input values. In particular, MLPs are neural networks structured in input, hidden and output layers, where each layer is fully connected with the following one. Among the different problems that deep learning solve, perhaps the most basic ones are regression and classification. A regression problem is a problem that tries to forecast a continuous value given an input (for example, predicting the price of a house given different characteristics of it). On the other hand, a classification problem is a problem that tries to assign a class to an input from a predefined set of classes (for example, predicting if an image contains a dog, a cat or a lion, like in our example). To allow a neural network to learn, the neuron functions depend on trainable parameters, which are parameters that are set optimizing (generally minimizing) a loss function. A loss function is a function that indicates how well a neural network models the function that solves a problem depending on a dataset, called training set if the dataset is used to train a neural network. In order to check if the model generalizes well for unseen inputs, Machine Learning practitioners use metrics like loss functions for regression problems or like accuracy ($\frac{\text{Number of examples predicted correctly}}{\text{Number of examples predicted}}$) for classification problems in different datasets than the ones used for training the networks, called test dataset.

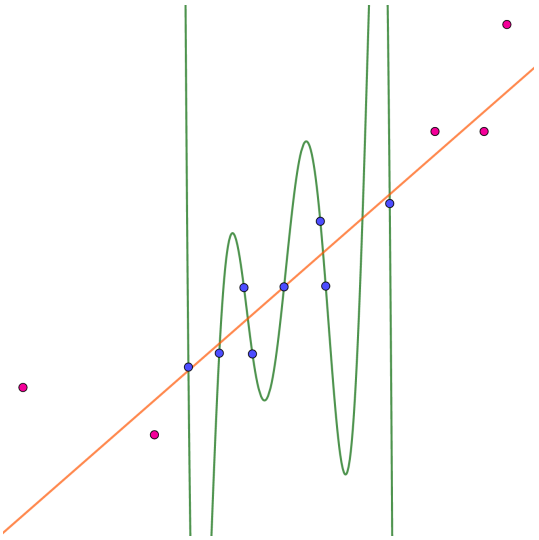


Figure 1.2: Purple and blue points are test and training datasets respectively in a regression problem modelling a function $f : \mathbb{R} \to \mathbb{R}$. The green curve and the orange line are, respectively, the interpolating polynomial and the linear regression of the training dataset points. In the image, although we can see that the green curve predicts perfectly the given values and the orange line has some prediction errors, when we compute the prediction error (sum of all the absolute differences between the predicted values and the actual values input) in the test dataset, we observe that the orange line has a lower value and thus we say that it generalizes better to unseen values than the green curve. In fact, the blue and purple points were extracted from the orange line adding some random noise to simulate data collection in a real environment. The generalization problem is the problem of recognizing when a model generalizes well or not, usually comparing models pairwise. Understanding generalization in deep learning is not an easy task to do because several internal representations of the data are learnt during the training phase whose complexity does not easily allow humans to interpret them. However, in some particular neural networks, especially in the first layers of them, there are studies dealing with information representation like [35].

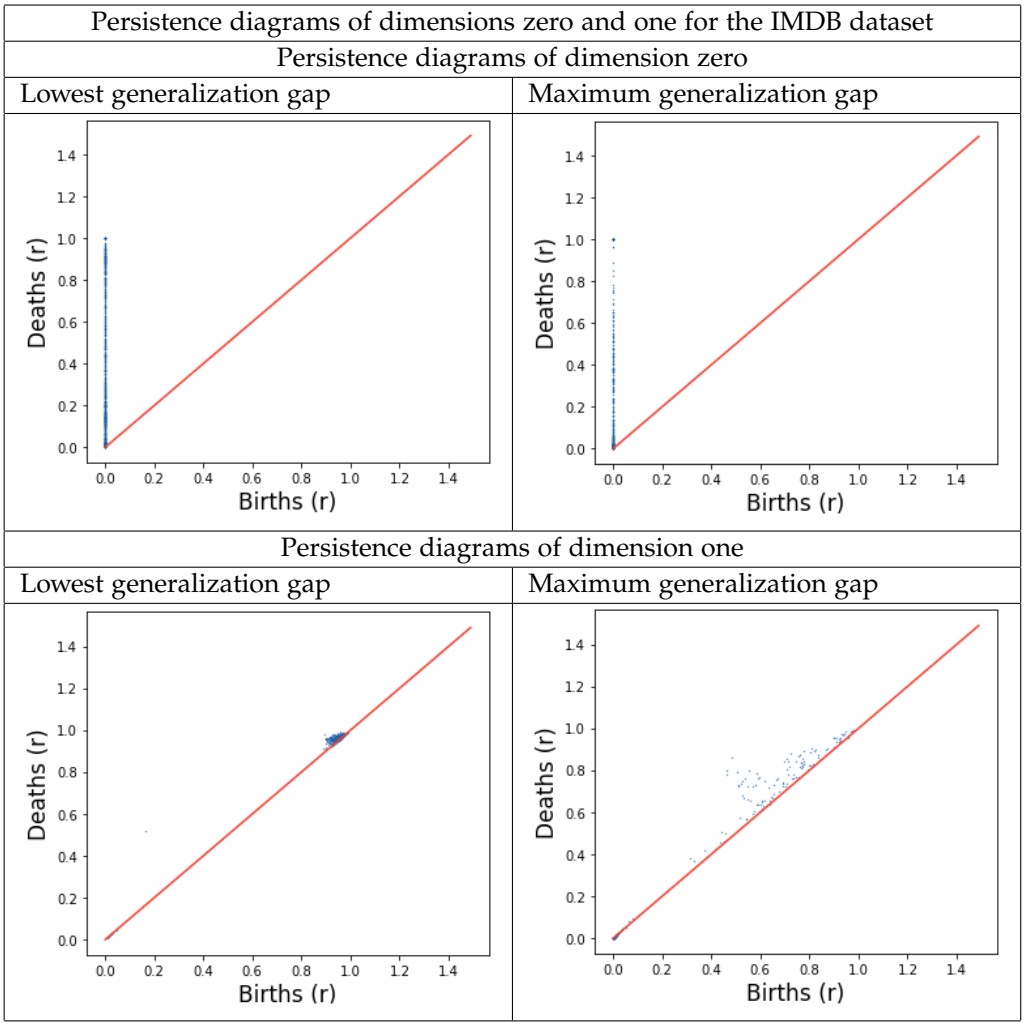


Figure 1.3: Persistence diagrams of dimensions 0 and 1 of the neural networks with lowest and highest generalization gaps (the higher the generalization gap of a model the worse this model generalizes, generally) of the IMDB dataset (one of the datasets we used in the experiment). Persistence diagrams are one of the main objects of study of topological data analysis. They are multisets (sets whose elements can have a multiplicity) of birth-death points (b, d) of a kind of objects called persistent classes. They summarize all the persistent homology information of our networks mapped to an space called activation graph, that is, a weighted nonnegative fully connected graph computed using a training dataset and a neural network. Persistent homology tracks shape features through a filtration, a mathematical object constructed, in our case, adding incrementally points of a weighted nonnegative fully connected graph. From persistence diagrams, that are not easily studied due to the complexity of their metrics, there are many topological features that can be built to study topological properties in an easier way like persistent entropy, average life or persistence landscapes. In the image it can be seen that persistence diagrams of low and high generalization gaps differ in both, quantity and distribution of points, so in this project we seek to find a relation between generalization and topological data analysis objects.

Chapter 2

Background theory

2.1 Deep learning

The objective of this project is to analyze how well a neural network generalizes using its intrinsic properties. To do that, we need to define what a neural network is and how we transform the usual neural network definition into a space where we can apply our topology methods. During the whole chapter we will be exploring these topics in a formal way, slightly different from other, more intuitive, definitions, in order to create a new theoretical framework to be used in the future.

But first, let us start by the basics. Let us define neural network step by step.

2.1.1 Computational graph

Definition 2.1. *A **computational graph** [9] is a 6-tuple*

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

where

1. *n is the number of vertices in the computational graph;*
2. *l is the number of in-degree leaves (from now on, leaves) in the computational graph such that $1 \leq l < n$;*
3. *E is the set of directed edges of the computational graph such that $\forall (i,j) \in E \quad j < i$ (that makes our graph **topologically ordered**), $j \in \{1,\dots,(n-1)\}$ and $i \in \{(l+1),\dots,n\}$;*
4. *each u^i, $i \in \{1,\dots,n\}$ is the **variable** associated with vertex i in the graph;*
5. *each d^i, $i \in \{1,\dots,n\}$ is the **dimensionality** for each variable, that is, $u^i \in \mathbb{R}^{d^i}$;*
6. *each f^i, $i \in \{(l+1),\dots,n\}$ is the **local function** for vertex i in the graph;*
7. *each α^i, for $i \in \{(l+1),\dots,n\}$, is defined as $\alpha^i := \langle u^j | (j,i) \in E \rangle$. This is a vector consisting on the vectors u^j concatenated.*

Defining $\overline{d^i} := \sum_{j:(j,i)\in E} d^j$, *then* $\alpha^i \in \mathbb{R}^{\overline{d^i}}$, $f^i : \mathbb{R}^{\overline{d^i}} \to \mathbb{R}^{d^i}$ *and* $f^i(\alpha^i) \in \mathbb{R}^{d^i}$

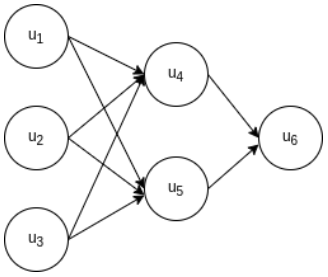


Figure 2.1: A simple computational graph representing an MLP with $n = 6$, $l = 3$, $E = \{(1,4),(2,4),(3,4),(1,5),(2,5),(3,5),(4,6),(5,6)\}$ with $d^i = 1$ for all i and f^4, f^5, f^6 the composition of a non-linear activation and a linear combination of the variables pointing in the graph to u^4, u^5 and u^6 respectively.

Each variable has two different functions. On one hand, they represent the formal nodes of the graph. Without any computation or algorithm, their values are initiated randomly and are meaningless. On the other hand, when leaf values $(u^1, ..., u^l)$ are provided and the forward algorithm is applied in these data, each variable contains the resulting value from applying the operations f^i sequentially, and they can be interpreted and used in other contexts, as we will be doing to compute topological features.

Many neural networks can be expressed using this mathematical model. In particular, MLPs, the kind of networks which we will be using through the analysis, can be clearly expressed as a computational graph naturally. Once we have our neural network expressed as a computational graph and a dataset of labelled examples, the next step is to map from our examples to actual values of the neurons (variables). As our graph is topologically ordered, we can define naturally a way to compute the value of each neuron (vertex) in the graph, using the algorithm known as **forward algorithm**.

Algorithm 1: The forward algorithm in computational graphs

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

output: u^n

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

Activation graph

One of the greatest concepts in [11] is the activation graph (called functional space in the paper). In it, we map from our computational graph to a weighted nonnegative fully connected graph where we will perform our topological analysis.

Definition 2.2. *Let X be a set. Let $m : X \times X \to \mathbb{R}^+ \cup \{0\}$ be a function. We say that m is a pseudo-semimetric if $m(x, y) = m(y, x)$.*

Definition 2.3. *Let $G = (V, E)$ be a weighted fully connected graph (i.e., a graph where for every pair of vertices there exists an edge connecting them with a weight). We say that G is a **weighted nonnegative fully connected graph** if the edge weights are nonnegative.*

Definition 2.4. *Let $\left(n, l, E, \{u^i\}_{i\in\{1,\dots,n\}\subseteq\mathbb{N}}, \{d^i\}_{i\in\{1,\dots,n\}\subseteq\mathbb{N}}, \{f^i\}_{i\in\{l+1,\dots,n\}\subseteq\mathbb{N}}\right)$ be a computational graph. Let $S = \{(x_i, y_i)\}_{i\in\{1,\dots,m\}}$ be a dataset where x_i is a vector of leaves that serves as input to the forward algorithm. Let $m : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ be a pseudo-semimetric on $\mathbb{R}^m$. Let $U_j = \{u^i_j\}$ be the set of u^i computed by the forward algorithm for the example x_j with $j \in \{1, \dots, m\}$ (not only the output u^n but all the u^i computed through its execution that are not leaves).*

*Defining u^i_j as the $u^i \in U_j$, we define the **neuron vector** as*

$$neur_i = (u^i_1, \dots, u^i_m)$$

*and $Neurons = \{neur_i\} \subseteq \mathbb{R}^m$. We call **activation graph** the weighted nonnegative fully connected graph (V, E) with $V = Neurons$ and*

$$E = \{(neur_i, neur_j, m(neur_i, neur_j)) : neur_i, neur_j \in V\}$$

all the possible edges between neuron vectors weighted by the value of the pseudo-semimetric applied pairwise.

Note that we have (at most, but in general exactly) $n - l$ points in our set *Neurons*, where each point has m coordinates. In our project, we work almost exclusively with a slight variation of the correlation function to get the pseudo-semimetric m (but in fact this is a hyperparameter and selecting the best one is an open problem). From this weighted nonnegative fully connected graph, we will be building filtrations of abstract simplicial complexes (we will define this concept later) to compute topological features. In fact, if we would not want to explain more theory about neural networks (how to train them, what a MLP is, etc.) we would be able to avoid more formalities and we would be able to explain directly algebraic topology.

As this is not our case, we want to keep constructing an intuition and a mathematical model of what a neural network is, so let us continue.

2.1.2 Deep learning basics

The objective of building neural networks is to model a function g that *approximates* a function f called latent model such that, given a dataset $S\{x_i, y_i\}_i$ where $x_i \in \mathbb{R}^l$ and $y_i \in \mathbb{R}^k$ (k is given by the problem), f is the true generator of y_i given x_i, i.e., $f(x_i) = y_i$. For example, assuming that a function f exists that determines whether an image is an animal or not, and assuming that we have a dataset of images labelled 0 if the image does not correspond to an animal and 1 otherwise, our computational graph (the function g), given an input x_i as leaves, will approximate this function f (classification problem). The (human) eye vision used to classify animals given an image could be catalogued as a function g that approximates f too.

But, how do we achieve these approximations? One way would be to give each function of the computational graph some parameters to learn. These parameters would be selected solving an **optimization problem**. Let us then define our optimization problem and how to solve it.

Definition 2.5. *Let $S = \{(x_i, y_i)\}_{i \in I \subseteq \mathbb{N}} \subseteq X \times Y$ be a set of points. Let $f : X \to Y$ be a function that tries to predict y_i from x_i ($f(x_i) = \hat{y}_i$). A **loss function** is a function $\lambda_f : X \times Y \to \mathbb{R}$ such that, the higher the value λ_f is, the higher the distances between the points y_i and $\hat{y}_i$ are overall. (Note that Y should have a metric.)*

If we fix the set S (training set) and we add some parameters $\theta_0, ..., \theta_n$ to the function $f(x)$ ($= f_{\theta_0,...,\theta_n}(x)$), then we can define the function $\lambda_{f,S}(\theta_0, ..., \theta_n)$ that varies depending only on the set of parameters chosen. This new function will be the one to be optimized. We will look for local (or global) minima in this new function to select the best hyperparameters for our function (expressed in our computational graph).

Examples of loss functions depend on the problem context. We can distinguish two basic types of inference problems: **regression** and **classification**. A regression problem is a problem that, given an input x_i, it tries to predict a continuous real value ($y_i \in \mathbb{R}$). A classical example is to predict the price of a house (y_i) given some of its properties (m^2, number of rooms, city/village in coordinates...) ($x_i \in \mathbb{R}^d$). On the other hand, a classification problem is a problem where, given an input, the function tries to identify a class in a set of possible classes that represents it. A classical example for this case would be recognition of handwritten digit images. Given an image input x_i (that is usually a tensor $n \times n \times 3$ where n is the number of pixels and 3 are the usual channels of color R, G and B), the function tries to predict which digit is written. See the MNIST problem [22] to see more about digit recognition.

For a regression problem, a typical loss function is the mean square error, that is defined as

$$\lambda_f(S) = \frac{1}{n} \sum_{i \in I} (y_i - f(x_i))^2.$$

For a classification problem, one of the most used loss functions is the cross-entropy loss, that for a binary classification, being $y = 0$ one class and $y = 1$ the other class, can be expressed as

$$\lambda_f(S) = -\sum_{i \in I} (y_i \cdot log(f(x_i) + (1 - y_i) \cdot log(1 - f(x_i)).$$

Now, we need to solve two more problems to close the chapter: What is the usual way to assign parameters to the functions f_i of the computational graph and how we optimize it. Let us start by solving the first one.

2.1.3 Multilayer perceptrons

Definition 2.6. *A **multilayer perceptron** is a computational graph defined by the 6-tuple*

$$\left(i, h, \{s^j\}_{j \in \{1,...,h\}}, o, E, \{f_\alpha\}_{\alpha \in \left\{ i+1,...,i+o+\sum_{j=1}^{h} s^j \right\}} \right)$$

where:

1. $i \in \mathbb{N}^+$ *is the dimension of the input data (i.e.,* $x_i \in \mathbb{R}^i$*);*
2. $h \in \mathbb{N}$ *is the number of hidden layers;*
3. $s^j \in \mathbb{N}^+$ *is the number of neurons in each hidden layer. The set of* s^j *can be the empty set if* $h = 0$*;*
4. $o \in \mathbb{N}^+$ *is the number of outputs (in regression* $o = 1$ *and in general classification* $o > 1$ *usually);*
5. *E is the set of edges of the computational graph. These edges must comply that input vertices can only be connected with the output if* $h = 0$ *or the first hidden layer if* $h > 0$*. Moreover, if* $h \neq 0$ *then the vertices of the hidden layer h can only be connected with vertices of the output layer. Finally, two neurons* a, b *of the hidden layers* h_1, h_2 *can be connected if and only if* $h_2 - h_1 = 1$ *with the connection* (a, b)*.*
6. f_α *are the functions of the computational graph.*

This 6-tuple generates the multilayer perceptron computational graph with $n = i + o + \sum_{j=1}^{h} s^j$, $l = i$, $d^1 = 1$ for all i. We will assign the vertices $1, ..., i$ to the input neurons (for an input $x_z \in \mathbb{R}^i$ we will take each coordinate value $x_{z,j}$ to be the value of the vertex j). Then, for the hidden layer t, we will assign the vertices from $i + \sum_{j=1}^{t-1} s^j + 1$ to $i + \sum_{j=1}^{t} s^j$. The remaining vertices will be assigned to the o output values. The set of edges is restricted as stated previously. The functions f_α are a linear combination of the values of the variables associated to the vertices connected with the corresponding variable $L : \mathbb{R}^{\#\text{number of neurons pointing to the variable}} \to \mathbb{R}$ combined with a nonlinear (**activation**) function $\phi : \mathbb{R} \to \mathbb{R}$ (i.e., $f_\alpha = \phi \circ L : \mathbb{R}^{\#\text{number of neurons pointing to the variable}} \to \mathbb{R}$).

This formal definition of multilayer perceptron is equivalent to the usual definition given in some books like [17] (Part II - Chapter 6), so we can save the values of the parameters of the linear combination in a matrix W, as usual (and how it is explained in the book). These parameters of the linear combination will be the ones that we will be optimizing in our loss function (luckily, in practice we will not be worrying about parameters and how to store them as TensorFlow abstracts these data representations when using Keras). The activation functions are hyperparameters that we will need to select. Among them, we can find the ReLU (Rectified Linear Unit) whose formula is $\phi(x) = max(0, x)$. We will be using this activation function during the whole project, but there are more like the Sigmoid one, the hyperbolic tangent, and so on.

Now we have a computational graph representing a complex function f with parameters (let us call them $\theta_1, ..., \theta_k$) to learn, so, taking one loss function $\lambda_f(S)$ and fixing a training set S we get the $\lambda_{f,S}$ to optimize. The question now is, how do we optimize this function? One first answer would be computing the local minima using the gradient of the function and taking the one that has the lowest loss value. However, in a complex scenario like the one we have, there is no easy way to do that (or even the gradient may not exist in a domain's subset, as it happens when we use the ReLU function as an activation function in the point zero) and we need to use **numerical optimization methods** to get

these minima in the best cases (or a subset of them) or points whose loss function values are low enough to stop the optimization process (usually, using iterative algorithms and stopping them after doing a fixed number of iterations, saving the best iteration values, not necessarily after a subset of minima has been found). However, a drawback of these methods is that we do not get always the lowest loss values, as these methods can get stuck in non-optimal local minima (except for **convex functions** where these methods converge to the global minima) or even not reach a minimum before the algorithm has stopped. However, there is no other option usually. Also, applying some numerical techniques, as **grid search** of hyperparameters for these numerical optimizers or early stopping (among several others), can yield nice results overall too without reaching global minima, as we have seen in recent applications of deep learning as autonomous driving or algorithmic trade, among others. These optimization approaches form an active area of research (in fact, a method has been published in October 2020 to use topology for optimization [32]).

For completion purposes, we will explain a simple numerical optimization method, the **gradient descent**, and we will connect it with the main algorithm to use it with computational graphs.

The gradient descent is a method to optimize the function $\lambda(x)$ that approaches global minima step by step, moving in the direction where the function decreases. To do that, gradient descent starts computing the **gradient of the function** $\nabla\lambda$ (usually numerically, avoiding the problem that a gradient is not defined in some isolated points giving a value for non defined derivatives) in an initial point a_0 (which is usually selected randomly). After that, the algorithm computes as the next point a_1 the value $a_0 - \alpha\nabla\lambda(a_0)$, where $\alpha \in \mathbb{R}^+$ is a small value called **learning rate**. Intuitively, this method tries to go against the direction of the gradient in each step (giving small steps, tuned using the learning rate α as the module of the gradient can be too big) because if the gradient in a direction is positive that means that the function increases in that direction, so we want to go against this direction. On the other case, if the gradient is negative, that means that in that direction the function decreases so, as we are subtracting a negative value, we advance towards that direction (and that for all the possible directions of the gradient). Doing this in every step, we get the formula $a_{n+1} = a_n - \alpha\nabla\lambda(a_n)$. Thus, we define the **gradient descent** method as follows:

Algorithm 2: The basic gradient descent

input : A C^1 function to optimize $\lambda(x)$, a learning rate $\alpha \in \mathbb{R}$, a number of steps to compute n and an initial point $a_0 \in Dom(\lambda)$

output: a_n

for $i \leftarrow 1$ **to** n **do**
  $a_i = a_{i-1} - \alpha\nabla\lambda(a_{i-1})$
end

There are several modifications to this algorithm like using always the same gradient for a number of steps, normalizing the gradient at each step, **stochastic gradient descent**... For more information on this topic the reference [17] can be consulted.

To compute the gradient of the loss function we can apply the **chain rule** to derive an expression depending on the gradient of the function f. In our case, the gradient of the function f is the gradient of the computational graph. How do we compute this gradient

then? This question is answered in [9] and the algorithm is called **backpropagation algorithm**. We will not explain it in detail, defining only local Jacobian functions and stating the algorithm, as [9] does a good job connecting computational graphs with the algorithm and explaining (and proving when necessary) all the steps and intuitions.

Definition 2.7. *For any edge $(j,i) \in E$ we define the **local Jacobian function** $J^{j\to i}$ as*

$$J^{j\to i}(\alpha^i) = \frac{\partial f^i(\alpha^i)}{\partial u^j} = \left.\frac{\partial u^i}{\partial u^j}\right|_{\alpha^i}^{f^i}$$

Let us now state the algorithm:

Algorithm 3: Backpropagation algorithm

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

output: For $j = 1, \dots, l$, output the Jacobian

$$P^j = \left.\frac{\partial u^n}{\partial u^j}\right|_{u^1,\dots,u^l}^{h^n}$$

for $j \leftarrow (n-1)$ **to** 1 **do**

$$P^j = \sum_{i:(j,i)\in E} P^i \times J^{j\to i}(\alpha^i)$$

(where α^i is calculated by the forward algorithm previously stated and $P^n = I(d^n)$ where $I(d^n)$ is the identity matrix of dimension d^n)

end

With this algorithm, we get the partial derivatives needed to construct the gradient of our function (or, if we compute the first derivatives numerically, an approximation of the partial derivatives, solving the problem of non-differentiable sets of points for some functions), and therefore we can execute a gradient descent process to let our neural networks learn.

2.2 Basic homology

2.2.1 Simplices

In this subsection, we will explain what a simplicial complex is and how to generalize this concept from affine geometry to a more abstract setting. For the whole section, we will use the notation used from [13].

Definition 2.8. *Let $S = \{u_0, u_1, \dots, u_k\}$ be points in $\mathbb{R}^d$. An **affine combination** of the set S is a point $x = \sum_{i=0}^k \lambda_i u_i$, $\lambda_i \in \mathbb{R}$, such that $\sum_{i=0}^k \lambda_i = 1$. The **affine hull** of S is the set of the affine combinations of S.*

Definition 2.9. *The points $u_0, u_1, ..., u_n$ are said to be **affinely independent** iff the n vectors $u_i - u_0$, $i = 1, ..., n$ are linearly independent.*

Definition 2.10. *An affine combination $x = \sum_{i=0}^{k} \lambda_i u_i$ of a set S is defined as a **convex combination** if $\lambda_i \geq 0$ for all i. The **convex hull** of S, denoted by $conv\{S\}$, is the set of convex combinations of S.*

Intuitively, if we draw the points of S in $\mathbb{R}^2$ and we enclose them by a rubber band, the convex hull represents all the points in the plane which are inside the band. See figure 2.2.

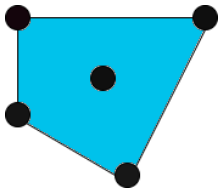


Figure 2.2: Five points (black circles) in $\mathbb{R}^2$. The convex hull is the blue surface plus the five points.

Definition 2.11. *A **k-simplex** is the convex hull $\sigma = conv\{S\}$ of a set S of $k+1$ affinely independent points.*

Definition 2.12. *The **dimension** of a k-simplex is k.*

The simplices with lowest dimension have their own names: 0-*simplices* are called **vertices**, 1-*simplices* are called **edges**, 2-*simplices* are called **triangles** and 3-*simplices* are called **tetrahedra**. In fact, the classical definitions of these figures coincide with the convex hulls of 1, 2, 3 and 4 points respectively.

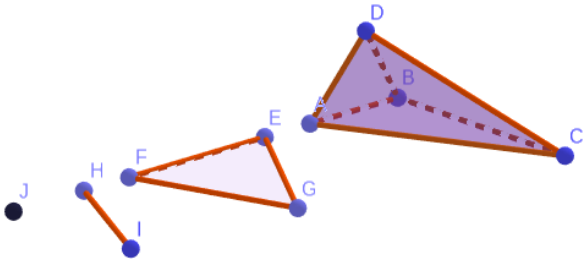


Figure 2.3: 0-simplex, 1-simplex, 2-simplex and 3-simplex coincide with the classical definitions of point, edge, triangle and tetrahedron respectively. Moreover, we can observe that every n-simplex is bounded by $(n-1)$-simplices. For example, triangles are bounded by three edges that are bounded, in turn, by two points per edge.

Note that every subset of a set of affinely independent points is affinely independent too. This means that every subset of points defines another simplex inside the original one. These simplices are called **faces** of the given simplex. We write $\tau \leq \sigma$ if τ is a face of

σ. The face τ is said to be **proper** if it is not σ ($\tau < \sigma$). If τ is a (proper) face of σ then we call σ a (proper) coface of τ.

2.2.2 Simplicial complexes

A simplicial complex is a set of simplices closed under taking faces and without improper intersections (intersections that are not simplices or empty sets). Formally,

Definition 2.13. *A **simplicial complex** is a finite collection of simplices K such that*

1. $\sigma \in K$ *and* $\tau \leq \sigma \Rightarrow \tau \in K$;
2. $\sigma_1, \sigma_2 \in K \Rightarrow \sigma_1 \cap \sigma_2$ *is a face of both* σ_1, σ_2, *or* $\sigma_1 \cap \sigma_2$ *is the empty set.*

Definition 2.14. *The **dimension** of a simplicial complex K is* $dim(K) = max_{\sigma \in K} dim(\sigma)$.

2.2.3 Abstract simplicial complexes

Up to now we have seen what a simplex is and how to construct simplicial complexes from them. The next step is to make abstraction of this concept beyond affine spaces, namely abstract simplicial complexes.

Definition 2.15. *A finite collection of sets A such that if* $\alpha \in A$ *and* $\zeta \subseteq \alpha$ *then* $\zeta \in A$ *is called an **abstract simplicial complex**.*

The sets in A are called **abstract simplices**. As we can see, the first requirement in the definition of a simplicial complex *coincides* with the requirement that abstract simplicial complexes have. The second requirement of the definition of a simplicial complex follows from the fact that the intersection between two of the sets of an abstract simplicial complex, if it is not empty, should be contained in both, and because of the only rule we have in this definition of an abstract simplicial complex then the intersection is included in A (note that the empty set is included in all sets).

The definition of dimension follows naturally from the definition of dimension in simplicial complexes and it is $dim(A) = max_{\alpha \in A}(|\alpha| - 1)$ where $|\cdot|$ denotes cardinality.

It is easy to see that we can create an abstract simplicial complex A from a simplicial complex K by retaining for each simplex only its vertices. The only fact that deserves a comment is that for each subset $\zeta \subseteq \alpha : \alpha \in A$, there exists an associated simplex with set of vertices equal to ζ. This follows from the fact that, by definition of convex hull, the simplex S associated with α contains all the possible convex combinations of its vertices. Hence, setting to 0 each scalar associated with a vertex not included in the subset ζ, we obtain the convex hull generated by the vertices in ζ. This is a face of the original simplex S. As this is a simplex, we have included by construction the set of its vertices in A, so the requirement to be an abstract simplicial complex holds.

2.2.4 Filtrations

Until now, we have stated what a simplicial complex is and how to make abstraction of this concept. As a consequence, we will be able to study simplicial properties (in

the future homological properties) for every topological space that admits a triangulation (generally transformed to an abstract simplicial complex, instead of being studied directly as a simplicial complex). However, *topological data analysis* usually deals with other structures such as point clouds (finite sets embedded in a metric space) or such as finite graphs instead of with well-known manifolds embedded in Euclidean space, either because the data represent other types of objects, like weighted graphs, or because the available data are observations from a source of truth (for example, point samples from a manifold) that is, *a priori*, unknown. In the latter case, these data do not have, *a priori*, any relationship between them except the origin of the observations (the source of truth) and ideally we would want to analyze the origin of the sample, but this is not possible in almost every situation.

To study weighted nonnegative fully connected graphs, it is crucial to develop a technique to originate and analyze simplicial complexes from them. But, which is the criterion to construct these new simplicial complexes? How are we going to deal with them? All the questions will be answered in this subsection.

First, we will define a way to construct simplicial complexes from weighted nonnegative fully connected graphs (see 2.3) whose edges are weighted by the values yielded by a pseudo-semimetric, as we did in 2.1.1. This construction will be useful to define a concept called filtration. Then, we will adapt our results to analyze.

Definition 2.16. *Let $r \in \mathbb{R}^+ \cup \{0\}$, and let $G = (V, E)$ be a weighted nonnegative fully connected graph. Let $\mathcal{P}(V)$ be the power set of V (the set of all possible subsets of V). Let $diam(\sigma) = max_{x,y\in\sigma} w(x, y)$, where $\sigma \in \mathcal{P}(V)$, be the diameter of σ according to the edge set E and $w(x, y)$ the weight of the edge connecting x and y (that exists because the graph is fully connected). The* ***Vietoris-Rips complex*** *of G with distance r is $VR_G(r) = \{\sigma \subseteq \mathcal{P}(V) : diam(\sigma) \leq 2r\}$.*

Varying the distance r leads the complex to have more or less abstract simplices. The bigger r is, the more abstract simplices are included in $VR_G(r)$.

To construct this simplicial complex, we check only pairs of vertices, adding them to the complex if their connecting edge weight is less or equal than $2r$. Then, we add all the higher-dimensional abstract simplices whenever all their edges are in the complex.

In general, we would want to compare weighted nonnegative fully connected graphs whose edge weights have been computed using the same pseudo-semimetric. In our case, we will compare different activation spaces (weighted nonnegative fully connected graphs; see 2.4) to extract generalization properties of deep neural networks given a pseudo-semimetric based on the correlation measure. To do that, it is especially interesting (and the usual way to proceed) to study a family of complexes regarding the r parameter, analyzing its evolution through different increments of r.

In order to define a structure for our simplicial complex families, we will define what a filtration is:

Definition 2.17. *A filtration is a family of sets $\{K_i\}_{i=0,...,m}$ such that $K_0 \subseteq K_1 \subseteq \cdots \subseteq K_m$.*

Now we will define the natural filtrations for our complexes:

Proposition 2.18. *Given a set $\{r_i\}_{i=0,...,m}$ where $r_i \leq r_{i+1}$ for $i \in \{0, 1, ..., m-1\}$ and a weighted nonnegative fully connected graph $G = (V, E)$, then $\{VR_G(r_i)\}_{i=0,...,m}$ is a filtration.*

This proposition follows clearly from the fact that for all $\sigma \subseteq S$, if $diam(\sigma) \leq r_i$ then $diam(\sigma) \leq r_{i+1}$ as $r_i \leq r_{i+1}$.

Now, we are ready to explain, first, classical homology (homology groups of abstract simplicial complexes) and later persistent homology (homology groups of filtrations).

2.2.5 Homology groups

Herbert Edelsbrunner and John L. Harer described homology groups in [13] as follows:

"Homology groups provide a mathematical language for the holes in a topological space".

And this is exactly what homology groups represent intuitively. This tool will allow us to define easy-to-compute features of an abstract simplicial complex and study them.

Definition 2.19. *Let K be a(n) (abstract) simplicial complex and $p \in \mathbb{N}$. A **p-chain** c is a formal linear combination of p-simplices in K with coefficients in a group G. If the p-simplices of the sum are $\sigma_1, ..., \sigma_n$ with coefficients $g_1, ..., g_n$ respectively, then we write $c = \sum_{i=1}^{n} g_i \sigma_i$. We denote the set of p-chains of K as $C_p(K)$.*

During the whole thesis, we will be using $G = \mathbb{Z}_2$ (group (in fact field) of $\mathbb{Z}$ modulo 2). This is important as the general definitions for some homological concepts we will define are adapted to this case and the generalization of them can differ.

In order to give the set of p-chains a group structure, we define the sum operation for p-chains:

$$\begin{aligned} +\colon & C_p(K) \times C_p(K) \to C_p(K) \\ & (c = \sum_{i=1}^{n} g_i \sigma_i, c' = \sum_{i=1}^{n} g'_i \sigma_i) \mapsto c + c' = \sum_{i=1}^{n} (g_i +_G g'_i) \sigma_i \end{aligned}$$

This sum is componentwise (like in polynomial sums). If σ_i is not in the formal sum of c' then $g'_i = 0$.

Proposition 2.20. *$(C_p(K), +)$ is a group. This group is called the **group of p-chains**.*

This proposition follows easily from the definition of p-chains and their sum operation.

We have now a group for each $p \in \mathbb{N}$. For $p > dim(K)$ and $p < 0$, $C_p(K)$ is the trivial group. To relate these groups, we will define the boundary operator for a(n) (abstract) p-simplex, and then we will apply it piecewise to linear combinations:

Definition 2.21. *Let $\sigma \in K$ be a(n) (abstract) p-simplex with vertices $\{u_0, u_1, ..., u_p\}$. Writing $\zeta = [u'_0, u'_1, ..., u'_n]$ for a(n) (abstract) n-simplex with vertices $\{u'_0, u'_1, ...u'_n\}$, we define the **boundary** of σ as the sum of its $(p-1)$-faces:*

$$\begin{aligned} \partial_p \colon & C_p(K) \to C_{p-1}(K) \\ & \sigma \mapsto \sum_{j=0}^{p} [u_0, ..., \hat{u_j}, ..., u_p] \end{aligned}$$

We can extend the definition of boundary from p-simplices to p-chains by applying the operation to every simplex in the chain, so we can define the **boundary** of a p-chain as: $\partial_p c = \sum_{i=1}^{n} g_i \partial_p \sigma_i$. In fact, this operation is a group homomorphism between $C_p(K)$ and $C_{p-1}(K)$.

With this operation, we can define chain complexes:

Definition 2.22. *A **chain complex** is a sequence of abelian groups connected by homomorphisms:*

$$\dots \xrightarrow{\partial_{p+2}} C_{p+1} \xrightarrow{\partial_{p+1}} C_p \xrightarrow{\partial_p} C_{p-1} \xrightarrow{\partial_{p-1}} \dots$$

such that $\partial_p \circ \partial_{p+1} = 0$ for all p.

In order to define homology groups, we will need two extra concepts and some preliminary results.

Definition 2.23. *A **p-cycle** is a p-chain c such that $\partial_p(c) = 0$. The subset of p-cycles is denoted as $Z_p(K)$.*

In fact, $Z_p(K)$ is a subgroup of C_p with the property that $Z_p(K) = Ker(\partial_p)$.

Definition 2.24. *A p-**boundary** is a p-chain that is the boundary of a $(p+1)$-chain.*

In a similar way, we have that the set of p-boundaries forms a subgroup of $C_p(K)$ (since the difference of two p-boundaries is the boundary of a $(p+1)$-chain that has all the non-common vertices of the two preimages of the boundaries). This group is denoted $B_p(K)$. In fact, $B_p(K) = Im(\partial_{p+1})$.

Now, we will give a fundamental result about the boundary operator on p-chains. Remember that we are working over $\mathbb{Z}_2$ but this result is valid in any p-chain group. Following the conventions of [12], we can find a proof in page 224.

Lemma 2.25. ***(Fundamental Lemma of Homology)**: $\partial_p \partial_{p+1} = 0$ for all $p \in \mathbb{N}$.*

Proof. As ∂_p is a homomorphism, we need to prove the result only for a single $(p+1)$-simplex. Let $\zeta \in C_{p+1}(K)$ be this simplex with vertices $u_0, ..., u_p$. By definition, $\partial_{p+1}(\zeta) = \sum_{j=0}^{p} [u_0, ..., \hat{u_j}, ..., u_p]$. Then, applying the boundary operator again, we have

$$\begin{aligned} \partial_p(\partial_{p+1}(\zeta)) &= \partial_p(\sum_{j=0}^{p} [u_0, ..., \hat{u_j}, ..., u_p]) = \sum_{j=0}^{p} \partial_p([u_0, ..., \hat{u_j}, ..., u_p]) \\ &= \sum_{j=0}^{p} \sum_{i=0, i\neq j}^{p} [u_0, ..., \hat{u_i}, ..., \hat{u_j}, ..., u_p]). \end{aligned} \tag{2.1}$$

Note that, for $i, j \in 0, ..., p$ such that $i \neq j$ every pair of vertices is removed from the generators exactly twice. Moreover, the sum has, for a sum of $(p+1)$ terms, a sum of p terms per term. Hence, the total sum contains $(p+1) \cdot p$ elements. Also, for p vertices the quantity of subsets of $(p-1)$ different vertices without order is $\binom{p+1}{p-1} = \frac{(p+1)!}{2!(p-1)!} = \frac{(p+1)p}{2}$. As each term of the sum is a $(p-1)$-simplex generated by a subset without two of the original vertices twice, and the quantity of these sets (because of the previous argument)

is $\binom{p+1}{p-1} = \frac{(p+1)p}{2}$, it follows that the sum consists of all the possible $(p-1)$-simplices (without order) generated by removing two different vertices from vertices generating ζ twice. This is

$$\sum_{i\neq j, i,j\in 0,\dots,p} 2 \cdot [u_0, ..., \hat{u_j}, ..., \hat{u_i}..., u_p]$$

As we are working with coefficients in $\mathbb{Z}_2$, we have that $2 = 0$ and hence the sum is a sum of zeros, proving the lemma. □

Corollary 2.26. *$B_p(K) \subseteq Z_p(K)$ and $B_p(K)$ is a subgroup of $Z_p(K)$.*

Now we are ready to define homology groups, the most important notion in this chapter:

Definition 2.27. *The **p-th homology group** of a(n) (abstract) simplicial complex K is the quotient*

$$H_p(K) = {}^{Z_p(K)}\!/_{B_p(K)}.$$

*Each element of the quotient group is called a **homology class**.*

Definition 2.28. *The **p-th Betti number** is the rank of $H_p(K)$ and it is written as $\beta_p(K) = rank(H_p(K))$.*

Many people visualize Betti numbers as the numbers of k-dimensional holes in a topological manifold (there are many examples but a relevant one is [18]). This intuition, according to Ethan D. Bolker in [4] follows from the concept of cycle. *"Intuitively, a k-cycle is a geometric structure which potentially surrounds a $(k+1)$-dimensional region. 1-cycles are easiest to visualize, think loops. [...] On the surface of a torus, there are two kinds of loops that don't have interiors, those are 1-cycles that are not boundaries, so a torus has 2 holes where there might be disks but aren't"*.

This intuition makes *easy* to understand the previous-defined concept to non-math users, so it is used in some state-of-the-art papers like the previous one.

2.3 Persistent homology

We have defined so far simplicial complexes, abstract simplicial complexes, filtrations and basic homology and p-chains. Let us now define the next step towards TDA, namely **persistent homology**.

In order to define what persistent homology is, we will follow the approach used in [28].

Let $\{K_i\}_{i=0,\dots,m}$ be a simplicial complex filtration. For each K_i we have its p-th homology group $H_p(K_i)$. Moreover, as $K_i \subseteq K_j$ for $i \leq j$, the inclusion maps $K_i \hookrightarrow K_j$ are well defined for $i \leq j$. These maps induce $\mathbb{Z}_2$-linear maps $f_p^{i,j} : H_p(K_i) \to H_p(K_j)$ with the property that

$$f_p^{k,j} \circ f_p^{i,k} = f_p^{i,j} \quad \forall i \leq k \leq j$$

because of commutativity of the diagram

$$\begin{array}{ccccccccccccc} \ldots & \xrightarrow{inc} & K_i & \xrightarrow{inc} & K_{i+1} & \xrightarrow{inc} & \ldots & \xrightarrow{inc} & K_{j-1} & \xrightarrow{inc} & K_j & \xrightarrow{inc} & \ldots \\ & & \downarrow & & \downarrow & & & & \downarrow & & \downarrow & & \\ \ldots & \xrightarrow{f_p^{i-1,i}} & H_p(K_i) & \xrightarrow{f_p^{i,i+1}} & H_p(K_{i+1}) & \xrightarrow{f_p^{i+1,i+2}} & \ldots & \xrightarrow{f_p^{j-2,j-1}} & H_p(K_{j-1}) & \xrightarrow{f_p^{j-1,j}} & H_p(K_j) & \xrightarrow{f_p^{j,j+1}} & \ldots \end{array} \tag{2.2}$$

Definition 2.29. *The **p-th persistent homology of** $\{K_i\}_{i=0,\ldots,m}$ is the pair*

$$(\{H_p(K_i)\}_{0\leq i\leq m}, \{f_p^{i,j}\}_{0\leq i\leq j\leq m}).$$

Definition 2.30. *The **p-th persistent homology groups** are the images of the homomorphisms induced by inclusion, $H_p^{i,j} = im(f_p^{i,j})$. Note that $H_p^{i,i} = H_p(K_i)$.*

Definition 2.31. *The **p-th persistent Betti numbers** are the ranks of the p-th homology groups, $\beta_p^{i,j} = rank(H_p^{i,j})$.*

According to [13], the persistent homology groups $H_p^{i,j}$ consist of the homology classes of K_i that are still *alive at K_j (the image of $f_p^{i,j}$ is not zero)*. For this reason, the following proposition holds:

Proposition 2.32. $H_p^{i,j} = {Z_p(K_i)}/{B_p(j) \cap Z_p(K_i)}$.

In our study of persistence, we will be interested in creation and destruction of homology classes through the filtration. In order to do that, we will define some useful concepts that we will be using later:

Definition 2.33. *Let $\zeta \in H_p(K_i)$. We say that ζ is **born** at K_i if $\zeta \notin H_p^{i-1,i}$ (intuitively, if ζ did not "exist" in $H_p(K_{i-1})$). Moreover, if ζ is born at K_i, we say that ζ **dies** entering K_j if $f_p^{i,j-1}(\zeta) \notin H_p^{i-1,j-1}$ but $f_p^{i,j}(\zeta) \in H_p^{i-1,j}$ (intuitively, if ζ merges with an older class, including the possibility of ζ "vanishing", as we go from K_{j-1} to K_j).*

Because of this definition, a homology class cannot be born and die in the same K_i.

Definition 2.34. *We define the **birth** of a homology class ζ as $b(\zeta) := i$ if ζ is born in K_i. Consequently, we define the **death** of a homology class ζ as $d(\zeta) := j$ if ζ dies entering K_j or $d(\zeta) := \infty$ if ζ never dies.*

Definition 2.35. *We define the **persistence** of a homology class ζ as $pers(\zeta) := d(\zeta) - b(\zeta)$. Note that, because of the previous clarification, $pers(\zeta) \neq 0$.*

Now we are going to define the most important object for our study, the persistence diagram, that will be used to compute different point cloud features to analyze our problem:

Definition 2.36. *We call **p-th persistence diagram** the multiset in the extended real plane* $\bar{\mathbb{R}}^2 = (\{\mathbb{R} \cup \pm\infty\})^2$ *of all the possible pairs* $(b(\zeta), d(\zeta))$*. This is equivalent to the multiset of the points* (i, j) *for* $i < j$ *with multiplicity*

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

Intuitively, the first difference of Betti numbers is equal to the number of generating classes born at or before K_i and dying in K_j. The second difference is equal to the number of generating classes that are born at or before K_{i-1} and die entering K_j. This means that $\mu_p^{i,j}$ counts the number of generators that are born at K_i and die entering K_j. If we draw this multiset, the persistence of a homology class born at K_i and dying in K_j is the length of the vertical segment from (i, j) to the diagonal $\{(x, x) : x \in \bar{\mathbb{R}}\}$ of the extended real plane.

The following property is the one that tells us that persistence diagrams encode all the information about persistent homology groups (and it is proved in [13]):

Lemma 2.37. ***(Fundamental Lemma of Persistent Homology)**: Let* $\{K_i\}_{i=0,\dots,m}$ *be a simplicial complex filtration. For every pair of indices* $0 \leq k \leq l \leq m$ *and every dimension* p*, the* p*-th persistent Betti number is* $\beta_p^{k,l} = \sum_{i \leq k} \sum_{j > l} \mu_p^{i,j}$*.*

Persistent homology on monotonic functions

We have seen a generic way to define persistent homology. However, we can extend the definition of persistent homology for functions $f : K \to \mathbb{R}$, where K is a simplicial complex, that are non-decreasing along increasing chains of faces, that is, $f(\sigma) \leq f(\zeta)$ whenever $\sigma \subseteq \zeta$ (such functions are called **monotonic**). These functions allow us to construct filtrations and to define everything we have defined in this chapter. This new approach allows an efficient computation of all the proposed metrics in this chapter.

Definition 2.38. *Let* $f : K \to \mathbb{R}$ *be a monotonic function where* K *is a simplicial complex. Let us call* $K(a) = f^{-1}(-\infty, a]$ *the **sublevel set** of the point* a*. Monotonicity of* f *implies that the sublevel set* $K(a)$ *is a subcomplex of* K *for all* $a \in \mathbb{R}$*. Given* $\{a_i\}_{i \in I \subseteq \mathbb{N}} \subseteq \mathbb{R}$ *with* $a_i \leq a_j$ *for* $i \leq j$*, we define the **monotonic filtration** of* f *with respect to* $\{a_i\}$ *as the filtration*

$$\cdots \subseteq K(a_{i-1}) \subseteq K(a_i) \subseteq K(a_{i+1}) \cdots$$

For these filtrations, we define exactly in the same way every concept we have defined (p-th persistent homology, p-th persistent homology groups, p-th persistent Betti numbers...) except the birth and death functions of a homology class. Let us define these values again for monotonic filtrations:

Definition 2.39. *Let* $\{a_i\}_{i \in I \subseteq \mathbb{N}} \subseteq \mathbb{R}$ *be a set defining a monotonic filtration* $\{K_i\}$ *of* $f : K \to \mathbb{R}$ *where* $K_i = K(a_i)$*. We define the **birth** of a homology class* ζ *as* $b(\zeta) := f(K_i)$ *if* ζ *is born in* K_i*. Accordingly, we define the **death** of a homology class* ζ *as* $d(\zeta) := f(K_j)$ *if* ζ *dies entering* K_j *or* $d(\zeta) := \infty$ *if* ζ *never dies.*

With these new definitions, we are going to change slightly the definition of persistence diagram for monotonic functions.

Definition 2.40. *Let $\{a_i\}_{i\in I\subseteq\mathbb{N}} \subseteq \mathbb{R}$ be a set defining a monotonic filtration $\{K_i\}$ for a monotonic function $f : K \to \mathbb{R}$ where $K_i = K(a_i)$. Let X be the original p-th persistence diagram of our monotonic filtration with the definitions of birth and death functions given in 2.39. The* ***p-th persistence diagram*** *for a monotonic function $f : K \to \mathbb{R}$ is the multiset $\{(f(K_i), f(K_j)) : (i,j) \in X\}$ and it is written $Dgm_p(f)$.*

As a final equivalence, we can get the Vietoris-Rips filtration in a weighted nonnegative fully connected graph $G = (V, E)$ whose edge weights have been computed with a pseudo-semimetric $m : V \times V \to \mathbb{R}$ for the set of radii $\{r_i\}_{i\in I\in\mathbb{N}}$, $r_i > 0$ and $r_i \geq r_j$ for $i \geq j \in I$ taking the diameter function for the pseudo-semimetric m (as we did in 2.16)

$$\begin{aligned} f\colon K &\to \mathbb{R} \\ c &\mapsto diam(c), \end{aligned}$$

where K is the simplicial complex that contains all the possible abstract simplices within the vertices of the graph V, that is clearly monotonic. In general, for a weighted nonnegative fully connected graph $G = (V, E)$ and an associated simplicial complex filtration $\{K_i\}_{i=0,\dots,m}$ we can move from the original definition (2.29) to this other one by taking as $f : K \to \mathbb{R}$, where $K = \bigcup_{i=0,\dots,m} K_i$, the value that assigns a simplex σ the index i of the first set in the original filtration such that $\sigma \in K_i$. Doing that assures us that the persistence diagrams and other definitions will coincide.

However, finding a nice monotonic function will lead to better results in other topics that we will not cover as, for example, estimating noise boundaries more accurately using stability theorems or comparing persistence diagrams using some distances that we will define soon.

In [3] we can find that persistence diagrams are computed using a Vietoris-Rips filtration from a *distance* matrix of points in a point cloud using this alternative definition (in our case the *distance* is the value yielded by a semi-metric). Specifically, this paper explains the background theory and algorithms used in the computation of persistence diagrams of the software we have used to compute them, Ripser.

From here, we will assume that we have computed a persistence diagram in order to go further.

2.3.1 Comparison of persistence diagrams

At this point we have defined an object from which we can extract useful information, the persistence diagram. However, how can we compare different persistence diagrams? To answer that, we will define two different metrics that give distances between persistence diagrams. From now on, we will work only with persistent homology of monotonic functions.

For this subsection, we will consider our persistence diagrams to have included in them infinitely many points on the diagonal, each with infinite multiplicity (i.e., if our persistence diagram was μ, we will work with $\hat{\mu} = \mu \cup (\bigcup_{i\in\mathbb{N}}\{(x,x) : x \in \mathbb{R}\})$ through the whole subsection.

Let us define the first distance function: the bottleneck distance.

Definition 2.41. *Let X, Y be two persistence diagrams. Let $B = \{\eta : X \to Y\}$ be the set of bijections between X and Y. The **bottleneck distance** between the persistence diagrams X and Y is*

$$W_\infty(X, Y) = inf_{\eta \in B}(sup_{x \in X}\|x - \eta(x)\|_\infty).$$

The bottleneck distance is a limit case (when $q \to \infty$) of the following family of distances.

Definition 2.42. *The **degree-q Wasserstein distance** between persistence diagrams X, Y is*

$$W_q(X, Y) = [inf_{\eta \in B} \sum_{x \in X} \|x - \eta(x)\|_\infty^q]^{\frac{1}{q}},$$

where B is the set of all the bijections between X and Y.

Wasserstein distances (including the bottleneck distance) have associated **stability theorems** relating closeness of persistence diagrams with proximity, in a suitable sense, of the data from which they come from. For more information on this topic, see [8].

2.3.2 Topological descriptors

From this section we will assume that we have generated a persistence diagram P from a weighted nonnegative fully connected graph $G = (V, E)$. From this point, we will need a way to compare different persistent diagrams and to extract properties of them. We have seen so far a way of comparing two persistence diagrams, that is computing their distance. However, using bottleneck and Wasserstein distances to perform some statistical and machine learning operations is cumbersome, as it is difficult even to perform basic operations as averaging (see [6]).

One way to relax the problems is mapping our persistence diagrams to a Hilbert space, in which we know several meaningful ways to compare elements (persistence landscapes and persistence images will be the representations we will be explaining for this approach).

Another way, even easier than the previous one, is to compute scores for each persistence diagram based on its points. This approach will lead us to define average life and midlife, among others.

Each approach has its pros and its cons. On one hand, having more sophisticated spaces can lead us to better metrics, but, on the other hand, the more capacity and complexity we have, the more prone we are to have overfitting in our methods (and this is solved using simpler descriptors, as in the second approach). From here, we will go trough each one theoretically.

Persistence landscapes

Intuitively, a persistence landscape is a function of persistence diagram points rotated 45 degrees clockwise. See figure 2.4.

There are several ways to define persistence landscapes. However, we will be using the one that it is defined from persistence diagrams in [6]. Note that, for this subsection,

we will work only with persistence diagrams which do not have coordinates placed at ∞. In our algorithms we will be discarding points (a, b) from persistence diagrams such that $a = -\infty$ or $b = \infty$. (However, these points may have useful information that we want to study and a way to analyze them is work to be done in the future.)

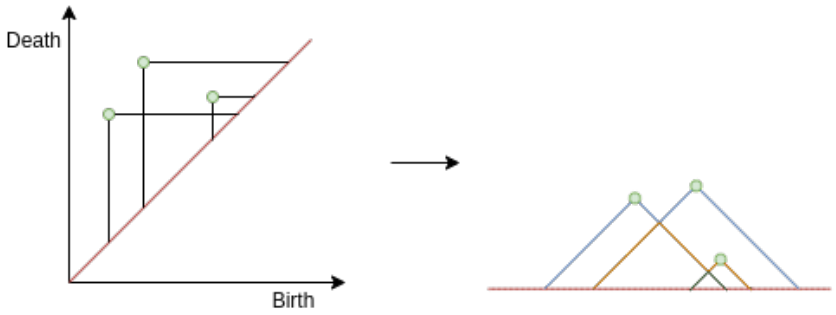


Figure 2.4: An arbitrary persistence diagram (left) and its landscape function image (colored curves). Each color curve represents a different k-persistence landscape. See 2.44.

Definition 2.43. *Let $\mathcal{V} = \{(a, b) \in \mathbb{R}^2 : a < b\}$. If we denote*

$$f_{(a,b)}(t) = max(0, min(t - a, b - t))$$

*for $(a, b) \in \mathcal{V}$, for all $t \in \mathbb{R}$, then the **persistence landscape** $\lambda : \mathbb{N} \times \mathbb{R} \to \mathbb{R}$ is the function*

$$\lambda(k, t) = \begin{cases} kmax_{(a_i,b_i)\in P-\{(x,y)\in P:x=y\}}(f_{(a_i,b_i)}(t)) & \text{if } k < |P - \{(x, y) \in P : x = y\}| \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

where $kmax(\cdot)$ is the k-maximum element function and $|\cdot|$ is the number of elements of a set.

Definition 2.44. *Given a fixed $k \in \mathbb{N}$, we call $\lambda_k(t) := \lambda(k, t)$ the **k-persistence landscape function**.*

Each k-persistence landscape function is piecewise linear with slope in $\{0, 1, -1\}$. The critical points of $\lambda_k(t)$ are the values of t at which the slope changes. A nice representation of this function computationally is the ordered set of critical points and critical values. With them, computing the area under these functions (and the kernel we will define later) is an easy (and fast) algorithm depending only on this finite set of points.

In our project, we will be using kernelized machine learning methods. Luckily, persistence landscapes allow us to give a kernel definition for persistence diagrams for these methods as the inner product of the Hilbert space were the persistence landscape lives. Let us first define what a kernel is:

Definition 2.45. *A **kernel function** on a set S is a symmetric map $K : S \times S \to \mathbb{R}$ such that*

$$\sum_{i,j=1}^{n} a_i a_j K(x_i x_j) \geq 0 \quad \forall\{a_1, ..., a_n\} \subseteq \mathbb{R} \quad \forall\{x_1, ..., x_n\} \subseteq S \quad \forall n \in \mathbb{N} : n \leq |S|.$$

As mentioned in [6], the space where persistence landscapes live is an L^2 space, specifically, $L^2(\mathbb{N} \times \mathbb{R})$. Therefore, we can associate a kernel function to the set of persistence diagrams, that is:

$$K(P^1, P^2) = \left\langle \lambda^1, \lambda^2 \right\rangle = \sum_{k=1}^{\infty} \int_{-\infty}^{\infty} \lambda_k^1(t) \cdot \lambda_k^2(t)\, dt,$$

where λ^1, λ^2 are the persistence landscapes of the persistence diagrams P^1, P^2 respectively.

Persistence images

Persistence images are visual representations of persistence diagrams. The author of [1] wanted to provide a finite-dimensional-vector representation with some nice properties as:

1. Stability respect to input noise.
2. Efficient computation.
3. Interpretable connection with persistence diagrams.
4. Flexibility when giving importance to concrete points in persistence diagrams.

In order to construct a persistence image from a persistence diagram, we will need to work a bit on some details first. As in persistence landscapes, we will suppose that we do not have points that have one of their coordinates equal to $\pm\infty$.

First, let $T : \mathbb{R}^2 \to \mathbb{R}^2$ be the linear transformation $T(x, y) := (x, y - x)$. This linear transformation converts the points of a persistence diagram in pairs with the birth and persistence of such point. Let $\phi_u : \mathbb{R}^2 \to \mathbb{R}$ be a differentiable probability distribution with mean $u = (u_x, u_y)$. This probability distribution will be a hyperparameter (a non-trainable parameter) when constructing the persistence images. However, during the whole project, we will be using the symmetric Gaussian function with mean u and variance σ^2 (another hyperparameter, this time easier to select). This function can be parameterized as:

$$g_u(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x-u_x)^2+(y-u_y)^2}{2\sigma^2}}.$$

Lastly, to finish our preparations, we will require to fix a nonnegative weighting function $f : \mathbb{R}^2 \to \mathbb{R}$ that is continuous, piecewise differentiable and zero along the horizontal axis $\{(x, 0) \in \mathbb{R}^2\}$.

In our project, we will be following the criteria used in [1] for experiments and we will use as f the function

$$f(x, y) = w_b(y) := \begin{cases} 0 & \text{if } t \leq 0 \\ \frac{t}{b} & \text{if } 0 < t < b \\ 1 & \text{if } t \geq b \end{cases} \tag{2.4}$$

where $b = max_{(b,d)\in(P)}(d - b)$ (maximum persistence of the points in the given persistence diagram).

Given all these definitions and hyperparameters, we can now map our persistence diagrams to a meaningful scalar map with the plane as domain.

Definition 2.46. *Let P be a persistence diagram. The corresponding **persistence surface** is the function $\rho_P : \mathbb{R}^2 \to \mathbb{R}$ defined as*

$$\rho_P(z) = \sum_{u \in T(P)} f(u)\phi_u(z).$$

In [1] it is proved that the weighting function f is critical to ensure that the map from persistence diagrams to persistent surfaces is *stable* (in the same sense of the stability theorems stated above; see Theorem 1 and Theorem 2 from [1]).

The persistence surface is the key component of the definition of persistence image.

Definition 2.47. *Let $R = \{r_1, r_2, ..., r_n\} \subseteq \mathbb{R}$, $C = \{c_1, c_2, ..., c_m\} \subseteq \mathbb{R}$ be ordered subsets. Define the **grid of R and C** as*

$$G(R, C) = \{s(r_i, r_{i+1}c_j, c_{j+1}) : \ i \in \{1, ..., n-1\}, \ j \in \{1, ..., m-1\}\} \subseteq \mathbb{R}^2$$

*where $s(a, b, c, d)$ is the convex hull of the points $\{(a, c), (a, d), (b, c), (b, d)\}$) (usually a square). Then, we define the **persistence image of the persistence diagram P with grid $G(R, C)$** as*

$$PI(P, G(R, C)) := \left\{ \int \int_p \rho_P \, dy \, dx : p \in G(R, C) \right\}.$$

Our persistence image can be visualized as the grayscale image with pixel values

$$pixel(i, j) = \int \int_{p=s(r_i, r_{i+1}c_j, c_{j+1})} \rho_P \, dy \, dx.$$

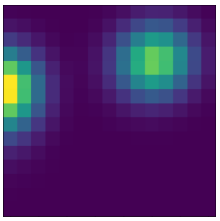


Figure 2.5: A persistence image generated from a sample from the NeurIPS generalization challenge dataset. As we can see, we have discretized the whole plane into a squared image, visualized following the formulae given in 2.47.

Easier features: Persistent entropy, average life, average midlife, persistence pooling and complex polynomials

Some good features (experimentally talking) are simpler ones. Averages of lives and midlives have given especially good results when predicting generalization gap using linear extrapolations. These definitions (except persistent entropy) have been used previously in the same problem context in [10]. Again, we will restrict our definitions to persistence diagrams without points (b, d) such that $b = -\infty$ or $d = \infty$.

Definition 2.48. *Given a persistence diagram P, we define its* ***average life*** *as*

$$\mathcal{L}(P) = \frac{1}{|P|} \sum_{(b,d)\in P} (d-b).$$

Definition 2.49. *Given a persistence diagram P, we define its* ***average midlife*** *as*

$$\mathcal{M}(P) = \frac{1}{|P|} \sum_{(b,d)\in P} \frac{d+b}{2}.$$

On one hand, average life is, arguably, the most basic scalar feature that one can extract from persistence diagrams. In fact, the concept of **life** is exactly the concept of persistence for a point in a persistence diagram. Therefore, when we use average life what we are really using is the average of persistences of a given diagram.

On the other hand, average midlife is just an average of midlives, that are other basic features that can be intuitively understood. The midlife of a point (b,d) in a persistence diagram is the point at the center of the interval (b,d), that is $b + \frac{d-b}{2} = \frac{b+d}{2}$. Midlives allow to take a representative point of an interval that measures, in contrast to lives which measure how much persistent classes live, the position in time where persistent classes lived (an analogy would be the age of people and the year in the middle of their births and deaths for life and midlife respectively).

These measures, although basic, are the foundation for more complex topological features, like **lifespan/midlife curves** [7].

Definition 2.50. *Given a persistence diagram P, we define its* ***persistent entropy*** *as*

$$\epsilon(P) = -\sum_{(b,d)\in P} \frac{d-b}{L} \cdot log_2\left(\frac{d-b}{L}\right)$$

where $L = \sum_{(b,d)\in P} (d-b)$.

Persistent entropy definition tries to adapt the concept of **entropy** originally discovered in the **information theory** field to topological data analysis. The original definition of entropy, given a discrete random variable X with probability distribution $p(x)$, is

$$H_X = -\sum_{x\in X} p(x) \cdot log_2 p(x) = \mathbb{E}\left(log_2\left(\frac{1}{p(X)}\right)\right).$$

According to [26], the entropy gives, intuitively, a measure of the uncertainty of the random variable. The smaller the entropy, the more *a priori* information one has on the variable's possible outcomes. In fact, sometimes entropy is called **missing information**.

Assuming, intuitively, that there exists a discrete random variable that distributes points in the persistence diagram such that the probability of a point (b,d) being in the persistence diagram is $p((b,d)) = \frac{d-b}{\sum_{(b,d)\in P}(d-b)}$, the persistent entropy formula follows. Moreover, if we assume that points with larger differences between their deaths and births give more information than points with smaller differences (in the same way other some works do like [15], where they consider points near to the diagonal to be noise), we

see that the higher the persistent entropy value is the more information it contains (or is useful).

There are several results regarding stability of persistent entropy that we will not state. For more information on persistent entropies, see [2].

Definition 2.51. *Given a persistence diagram P, the **persistence pooling vector** of n values is the vector in descending order of the n maximum persistence values in P. If the persistence diagram has i points and $i < n$ then the vector has the first i values equal to the i persistence values of the persistence diagram in descending order and the other values are 0 (intuitively, if a point does not exist, it has persistence equal to 0). Rigorously, the persistence pooling vector $V(P) = (v_1, v_2, ..., v_n)$ has values*

$$v_i = \begin{cases} imax_{(b,d)\in P}(d-b) & \text{if } i \leq |P| \\ 0 & \text{otherwise,} \end{cases} \tag{2.5}$$

for i in $1, ..., n$, where $|P|$ is the number of points of the persistence diagram P and $imax(\cdot)$ is the i-th maximum value.

Persistence pooling vectors were introduced in [5] in order to improve the max-pooling they were doing to solve a problem using TDA. This method rely on the information given by the biggest persistences, assuming that more persistent features give more information than less persistent ones (like we did in the section 2.50).

Persistence pooling vectors have their own stability results. For further information, we refer the reader to [5].

Definition 2.52. *Let P be a persistence diagram. Let $\phi : P \to \mathbb{C}$ be a function that transforms points from the persistence diagram into points in $\mathbb{C}$ (we will see some of them later). Let*

$$p_\phi(x) = \prod_{(b,d)\in P} (x - \phi((b,d))) = x^{|P|} + \sum_{i=0}^{|P|-1} a_i x^i,$$

*where $|P|$ is the number of points of the persistence diagram P, be the minimal degree monic polynomial whose roots are the transformed points in $\mathbb{C}$ of the persistence diagram by ϕ. The **vector of the n leading coefficients of the complex polynomial generated from $\phi(P)$** is the vector $CP_n = (r_1, i_1, ..., r_n, i_n)$ where $r_j = Re(a_{|P|-j})$, $i_j = Im(a_{|P|-j})$ for $j \in \{1, ..., n\}$ if $|P| - j \geq 0$ and $r_j = 0$, $i_j = 0$ otherwise.*

The idea of transforming persistence diagrams points into complex numbers to work with them as a roots of a complex polynomial was introduced in [14]. There, three different ϕ functions, R, S and T, were introduced and defined as follows:

1. $R(u, v) = u + iv$.
2. $S(u, v) = \begin{cases} \frac{v-u}{\alpha\sqrt{2}}(u + iv) & \text{if } (u, v) \neq 0 \\ (0, 0) & \text{otherwise.} \end{cases}$
3. $T(u, v) = \frac{v-u}{2}(cos\alpha - sin\alpha + i(cos\alpha + sin\alpha))$, where $\alpha = \sqrt{u^2 + v^2}$.

R, S and T are continuous maps where R and S are also injective. Note that R is the identity function, S modifies slightly the position of the identity function, sending points near the diagonal near to 0 but respecting $Re(S((b,d))) < Im(S(b,d))$ and, finally, T distributes the points across the $\mathbb{C}$ plane sending points near to the diagonal to a neighborhood of 0 again (but it is not injective).

Due to the Viète formulas, we can compute easily the coefficients of the monic polynomial $p_\phi(x)$. Moreover, due to their continuity, close roots imply close coefficients and therefore close vector representations.

One key question before using this technique is why we are using and comparing vectors that represent coefficients of different degrees in different polynomials. The experimental results obtained in [14] showed that the coefficients of higher degree terms were more meaningful than the ones with lower degrees. In fact, it was shown that the bigger n was, the less difference between the results for n and $n-1$ existed.

Also, it was shown that coefficients of complex polynomials had performed as good as bottleneck distances and, in some cases, even better. For more information on this topic, see [14].

In general, these kind of polynomials computed from points in a persistence diagram open a new research area about how to take profit of these polynomials in general (not only as vectors of coefficients but also as holomorphic functions).

Chapter 3

Analysis, design and implementation

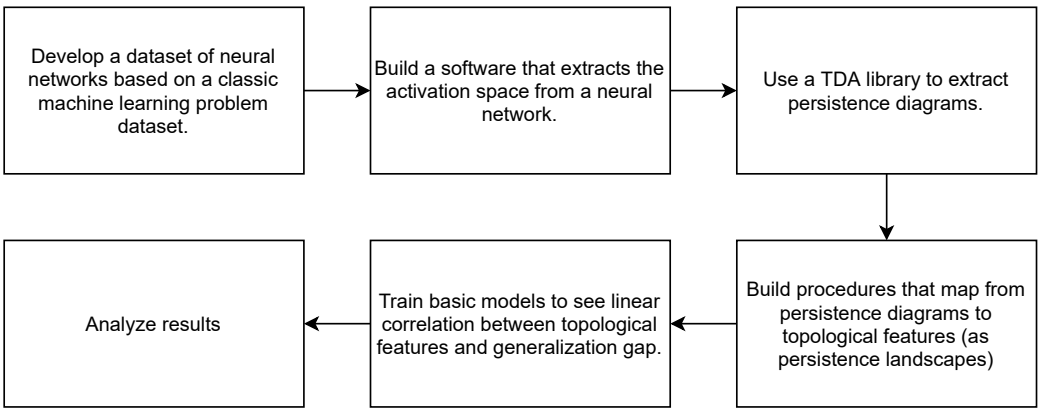


Figure 3.1: Basic pipeline of the project.

We have seen so far all the theory we need to develop the whole project. Our objective in this experiment is to link the generalization gap with our topological features. The generalization gap is a formula that relates how nice a machine learning model performs in a classification problem in the set used for training the network and in new, unseen, data (test dataset). The formula of the generalization gap is

$$g(\hat{f}, \mathcal{D}) := \frac{1}{|\mathcal{D}_{train}|} \sum_{(x,y)\in\mathcal{D}_{train}} \mathbb{1}(\hat{f}(x) = y) - \frac{1}{|\mathcal{D}_{test}|} \sum_{(x,y)\in\mathcal{D}_{test}} \mathbb{1}(\hat{f}(x) = y)$$

where D is a dataset, split in two datasets called **train dataset** (60% of the dataset) and **test dataset** (20% of the dataset). The other 20% of values is inside a third set called **validation dataset** used for selecting the best hyperparameters in the generation of the neural network's dataset phase.

The pipeline we have followed for the development of this project has been the one shown in 3.1.

3.1 Development of a dataset

For the experiment analysis we need a dataset where we can try our topological tools. As we are trying to analyze generalization of neural networks given a specific dataset, **our** dataset will consist on a classical dataset (like MNIST) splitted in training, validation and testing datasets and many different neural networks trained on the previous training and validation datasets. The generalization gap will be computed using the training and testing datasets.

Firstly, we used the dataset given by [19], where several complex neural networks were given together with the problem datasets used to train them split in training and test subdatasets (we did not need the validation dataset as the networks were already trained). However, the complexity of the networks given was impossible to handle with the tools we developed, as we had thousands of neurons per each network and thousands of examples in each problem dataset. The computational tools we used for computing persistence diagrams, first **dipha** and later **Ripser**, were incapable of processing such quantity of data (Ripser starts to get stuck at 1000 points, as mentioned in [29]). For this reason, we tried to subsample the networks' neurons that were used for processing persistence diagrams with the prior software in an attempt to extract useful information from a subset of the data. However, we saw experimentally that the quantity of data sampled was not enough to achieve significant results. As a consequence, we decided to relax the problem and to build simpler neural networks with a maximum of 1000 neurons per network.

To build these networks, we selected two different classical problems to analyze, the **CIFAR10** classification (image classification) [21] that includes images that are split in ten different categories, and the **IMDB reviews** sentiment classification (text classification) [25], that includes over 50000 movie reviews associated with a binary sentiment polarity label (positive review or negative review).

We selected two different-nature problems instead of only one to see how topological descriptors evolve when the problem domain changes and to see if there are unified features that do not depend on the problem dataset.

Initially, each dataset included only training and tests subdatasets (when downloaded). However, we split each one in 3 subdatasets, the training dataset, that was composed of 80% of the data included in the training folder, used for training the networks, the validation dataset, used for selecting the best hyperparamaters during the training phase, consisting on the remaining 20% of the data in the training folder, and the test dataset, used for validating our results in our predictions of the generalization gap. Each subdataset was split in batches of 32 elements in turn.

Regarding preprocessing of the datasets, we did not preprocess the CIFAR10 dataset images, as they were ready to go due to the dataset was retrieved using the Tensorflow API. For the IMDB reviews dataset we normalized the texts (symbols deletion, xml tags removal and characters transformation in underscore), removed the stopwords enumerated in `nltk.corpus` module and lemmatized the resulting words using **nltk** (the code

can be seen at `ToyNeuralNetworks/datasets/imdb_reviews/preprocess.py`).

In both problems, we generated 10 synthetic complete MLP networks approximating the shape of a **Pareto distribution** (see figure 3.2). For each network generated as in the figure, we saved as different variations the evolution of the network per each new hidden layer, that is, if the neural network had n hidden layers (3 in the figure), we would save sequentially the neural network with the input, the first hidden layer and the output. Then we would save the previous network but with the second hidden layer added just before the output layer and so on until the whole network of n hidden layers was saved. (The code is included in the file `ToyNeuralNetworks/datasets/networks/MLP/mlp_generation.py`.) Doing this process, and limiting the number of hidden layers to 6, we generated finally 60 different neural networks for each problem.

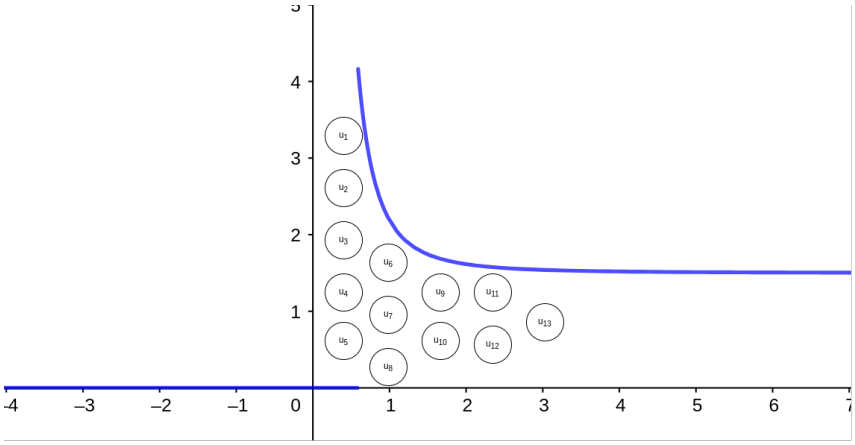


Figure 3.2: Neurons of a fully connected MLP following a Pareto distribution shape. The Pareto distribution density function of parameters α and σ is defined as

$$f_{\alpha,\sigma}(x) = \begin{cases} \alpha \frac{\sigma^\alpha}{x^{\alpha+1}}, & \text{if } x > \sigma \\ 0 & \text{otherwise.} \end{cases}$$

Adding a third hyperparameter $\delta > 0$ to the function such that $f_{\alpha,\sigma,\delta}(x) = \delta + \alpha \frac{\sigma^\alpha}{x^{\alpha+1}}$, if $x > \sigma$ we get a flexible enough shape with some parameters to modify to obtain different neural network graphs.

Once data preprocessing was finished we trained the 60 networks per problem [1]. We wanted to analyze not only fully nice-fitted networks but also we wanted to add to our analysis networks in the three well-known states of a machine learning model, the **underfitted** state (a version of the network that had not learnt enough from the dataset, having bad performance both in training and test data), the **nice fitted** state (a version where the network had learnt from the data without memorizing, that is, that it had nice performance both in training and test datasets) and finally the **overfitted** state (a version of the network that had memorized the training data and performed exceptionally good on

[1] The procedures used to train the networks can be found in `ToyNeuralNetworks/datasets/<CIFAR10|imdb_reviews>/train.py`.

training data but poorly on test data). We wanted that to add variability to our study and to see if topological properties were good enough to classify in a variety of typical situations. For this reason, we saved the model in each iteration of the training process (trying to capture iterations in these 3 different states). In order to avoid going too fast to the overfitted region of the training process and therefore not being able to select an underfitted example (this actually happened during the experiment), we trained the model following a **Fibonacci training sequence**. This means doing each iteration of the training with a $max(\text{number of elements in an epoch}, a_n)$ batches of data where a_n is the a_n element of the Fibonacci sequence and n the iteration number (note that a_0 does not necessarily start at 1; in fact, it is possible to set an arbitrary number as starting iteration value configuring the `Fibonacci` function parameters). Doing that, we were able to analyze many different iterations to select the iterations that best described these states.

The training approach was standard: we used the `SparseCategoricalCrossentropy` and `BinaryCrossentropy` losses from the **Keras** API for CIFAR10 and IMDB reviews respectively. As optimizer, we selected the `Adam` optimizer from Keras with standard parameters (because it was the one that performed best with our datasets experimentally, although is not explained in the background theory chapter because of we did not want to deepen in the optimization topic). In both datasets, we did a pretraining of 4 epochs with several learning rates in the set ($\{0.01, 0.001, 0.0001, 0.00001, 0.000001\}$) to select the one that fitted best the model (according to the validation dataset). Once the best learning rate was selected, we performed a training of 150 epochs per each network.

Once the networks were trained, we analyzed the training iterations to automatically select the representative iterations of the 3 states (note that we trained 60 networks and we needed a way to filter them as doing it one by one would have been untractable, due to the quantity of times we needed to repeat the experiment). Each model was slightly different, so we will analyze both individually. [2]

3.1.1 IMDB reviews iterations choice

Defining rigorously underfitting, nice fitting and overfitting is not an easy thing. One way to do that is to see learning and validation losses of our model through the learning phase (we saved their values when training the network as we did with iterations) and decide intuitively which iterations work best with new data. Generally, we say that a model is nice fitted at the iteration i_{nice} when the distance between training and validation losses is minimum (and both are small enough). Following intuitions, we can say that the model is overfitting after the nice iteration when the losses start to separate again (and thus, memorizing), increasing the value of the validation loss and decreasing the value of the training loss (or remaining more or less equal. In an ideal case, we do not really know which will be the behaviour of the losses in every model). Finally, an underfitted iteration will be one of the first iterations of the training process, as it did not have time to learn enough from the examples (where both losses are high and, more or less, distant).

According to our descriptions, when selecting underfitted, nice fitted and overfitted iterations, we seek to base our decision on the global behaviour of the losses. However,

[2] Both analyses can be executed with the data using the Jupyter Notebook `ToyNeuralNetworks/playground-notebooks/AnaylzingDataToyNetworks.ipynb`.

this could be difficult when using directly the loss functions. Instead, we will be using smooth functions with global minima that are good enough approximating the losses. In particular, we seek to obtain functions with only one local minimum that is also the global minimum.

After some tries, we could see that, in general for all the models trained with the IMDB reviews, the losses could be approximated with a function of the type $\hat{l}_{a,b,c}(x) = a \cdot \frac{1}{x} + b \cdot log(x) + c$ where a, b, c are hyperparameters trained using a linear regression. Computing the R^2 score (where a value of 1 means that the model of the data is linear) [31] for each model we extract the following statistics:

1. Min R^2 score for extrapolations for validation loss curves: 0.267
2. Max R^2 score for extrapolations for validation loss curves: 0.896
3. Average R^2 score for extrapolations for validation loss curves: 0.589
4. Min R^2 score for extrapolations for training loss curves: 0.0421
5. Max R^2 score for extrapolations for training loss curves: 0.946
6. Average R^2 score for extrapolations for training loss curves: 0.714

This means that, in average, 58% and 71% of the variability of the dependent variable is explained with this model for the validation loss and training loss respectively. See figure 3.3.

With this extrapolated function, we selected for each network as nice fitting i_{nice} the iteration with less distance between losses. As overfitting iteration the iteration $i_{overfitted} := i_{nice} + 100$. Lastly, as underfitting we selected the iteration i_{under} such that

$$i_{under} = argmin_{i \in \text{Iterations}} \left(\left| \frac{loss_{val}(i_0) - loss_{val}(i_{nice})}{2} \right| \right)$$

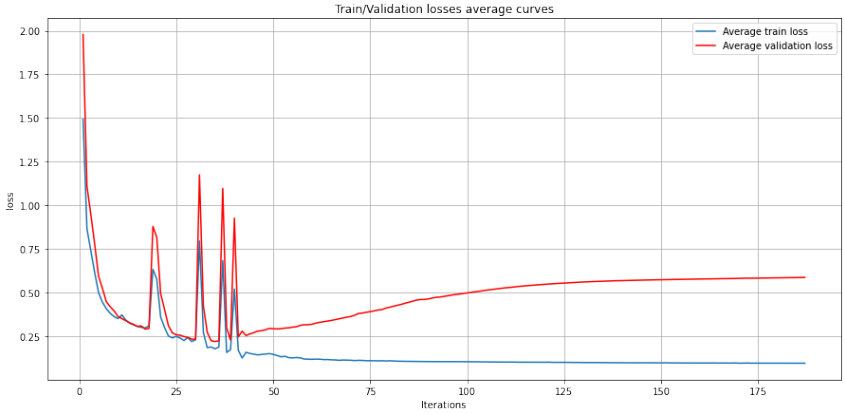
(we did not take the first iteration because at least we want a network that has learnt something and whose parameters were not just randomly initialized). [3]

3.1.2 CIFAR10 iterations choice

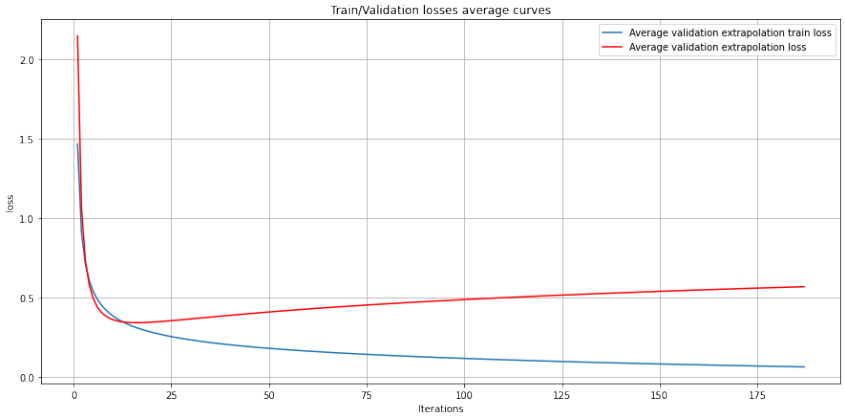
For CIFAR10 we had even better results using a quadratic model $\hat{l}_{a,b,c}(x) = a \cdot x^2 + b \cdot x + c$. In this particular case, we obtained the following data:

1. Min R^2 score for extrapolations for validation loss curves: 0.007
2. Max R^2 score for extrapolations for validation loss curves: 0.99
3. Average R^2 score for extrapolations for validation loss curves: 0.90
4. Min R^2 score for extrapolations for training loss curves: 0.004
5. Max R^2 score for extrapolations for training loss curves: 0.980

[3] The code is located in `ToyNeuralNetworks/Networks/pick_inverted.py`.

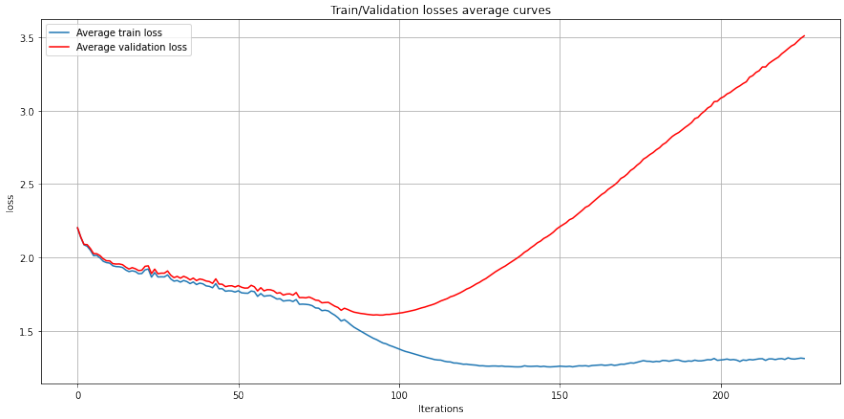


(a) Non extrapolated losses.

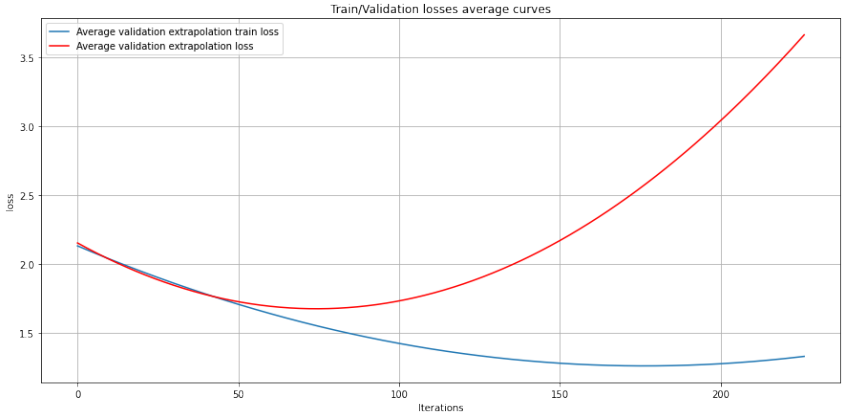


(b) Extrapolated losses with our model

Figure 3.3: Train and validation average losses of all the IMDB reviews networks during the training process. The x-axis indicates the iteration number where the y-axis indicates the average loss in the corresponding iteration. Validation and training losses are painted in red and blue respectively. As we can see, the model can be nicely extrapolated to a model of the form $\hat{l}_{a,b,c}(x) = a \cdot \frac{1}{x} + b \cdot log(x) + c$, as it is shown in the figure *(b)* to describe the general behaviour of the loss functions.



(a) Non extrapolated losses.



(b) Extrapolated losses with our model

Figure 3.4: Train and validation average losses of all the CIFAR reviews networks during the training process. The x-axis indicates the iteration number where the y-axis indicates the average loss in the corresponding iteration. Validation and training losses are painted in red and blue respectively. As we can see, the model can be nicely extrapolated to a model of the form $\hat{l}_{a,b,c}(x) = a \cdot x^2 + b \cdot x + c$, as it is shown in the figure *(b)* to describe the general behaviour of the loss functions.

6. Average R^2 score for extrapolations for training loss curves: 0.875

With this model, we applied a more sophisticated algorithm to choose the nice fitting representative iteration and we used the previous heuristic selections for the overfitted and underfitted ones. This newer algorithm for selecting the nice fitting iteration consisted on taking the intersection between two losses after the validation loss minimum (that resulted, in this experiment, to be always a convex function). If this intersection had not existed, we would have taken the same iteration as we took in the IMDB reviews problem. The code can be found in `ToyNeuralNetworks/Networks/pick.py`).

Note that there were some degenerate cases for both datasets (for example, when the

extrapolated model had an R^2 of 0.007 in the CIFAR10 problems). In these cases, that are less than 5 per model, an underfitted, overfitted or nice fitted iterations could not be found. For this reason, the amount of networks for each classical problem in the following chapters are slightly less than 180, but in any case we had a similar amount of networks for the 3 states).

3.2 Computation of persistence diagrams

Recapping, we have two different classical problems, CIFAR10 and IMDB reviews sentiment analysis. Each problem has about 180 models (60 different models in 3 different states, that count as different networks in our study as they do not have the same parameters (thus they do not have the same activation graph nor persistence diagrams) and give different outputs).

The code of this section is developed in the project *NeurIPSSoftware*. In it, the entry point of the software is the Python script `ToyDatasetPDComputation.py`. This script was used to generate the persistence diagrams in the CVC (Computer Vision Center) server using the docker image and configuration found at the folders `devops` and `docker-image`. The explanation of the persistence diagram computation is very easy but the technical details are complex, so feel free to check the code to contrast every step stated in the explanation of the subsection. Note that we are using software that implicitly uses the definitions of the background theory. Recall that Tensorflow, in its first version, used directly the computational graph representation, and nowadays it can be retrieved, as we will show in the next paragraphs.

Once the networks were trained and selected, the next step was to obtain the activation graph from neural networks. Tensorflow, the framework we used in the project, admits the exploration of layers of an MLP, letting the user to use the function of each neuron given an appropriate input. In this way, doing something like:

```
...
model = get_tensorflow_model()
X_train = get_X_train()
x = X_train
layer_values = []
for layer in model.layers:
    x = layer(x)
    layer_values.append(x[:])
...
```

we retrieved a list of layer values of each network for all the values in the tensor X_train (where the first dim contains the examples, listed in chunks of 32 examples, because of the split we did generating the datasets). This feature allowed us, after some refinement of the data structure, to reconstruct the computational graph and to compute the activation graph (see 2.4), using the following pseudo-semimetric, modified from the correlation one:

$$m_X(x,y) = 1 - |corr_X(x,y)|,$$

where $corr_X(x, y)$ is the correlation of the points x and y in the set X (see [27]) whenever it can be computed and 0 otherwise (the correlation cannot be computed only when one of the variables is constant. Hence, it can be considered that the correlation is zero with any variable because firstly, intuitively, we can say that a constant distribution is not correlated with any other distribution as it is independent from them and secondly, because we need to use this convention to let m be a pseudo-semimetric. More information on this decision can be found in [34]).

This pseudo-semimetric gives us, intuitively, that, the higher correlation exists between two neurons, the *nearer* they are in the graph.

Once the activation graph was computed, we generated a matrix distance of all the points in the activation graph. This matrix was the input of **Ripser** [3]. Ripser is the tool that has the most efficient algorithm to compute persistence diagrams, according to [28]. With it, we computed the 0, 1 and 2-persistence diagrams for each neural network (we tried more dimensions but experimentally the dimensions that gave more information about the neural networks were 0, 1 and 2).

3.3 Predicting generalization gap using topological features

In this section we will explain how we computed topological features to train linear models to predict the generalization gap of each network. The experiments of this section were executed in three different datasets, the IMDB reviews dataset and the CIFAR10 dataset plus a fused dataset containing the persistence diagrams and generalization gaps of both problems to see if there existed topological features that yielded models robust to context changes. Note that the y values of our dataset (the labels for each neural network in our two datasets) were the generalization gaps of each model and they were not computed yet. They were generated precisely at this stage because we did not need them previously. The generalization gaps were computed in the file `NeurIPSSoftware/ComputeMetricsForDataset.py` for both datasets. Note that now the X values of our datasets are the corresponding persistence diagrams for each network model (about 180 persistence diagrams per dataset in the individual datasets, IMDB reviews and CIFAR10, and about 360 persistence diagrams in the fused dataset). Due to the correspondence between neural networks and their persistence diagrams, we will use neural networks and persistence diagrams interchangeably, although in reality, at this, step, we use only persistence diagrams.

To compute linear regressions, we minimized two different loss functions (using `Scikit learn`), the ordinary **least squares** loss, that is, given a dataset $X_{train} = \{(x_i, y_i)\}_{i\in\{1,\dots,n\}}$ with $x \in \mathbb{R}^d$ and a linear model to fit $f_\beta(x) = \sum_{j=1}^{d} \beta_j \cdot x_j + \beta_{d+1}$ with parameters β in $\mathbb{R}^{d+1}$,

$$LS(X_{train}) = \sum_{i=1}^{n} (y_i - f_\beta(x_i))$$

and the **Ridge linear loss** that is the linear least squares with an ℓ_2-norm regularization parameter over the parameters β depending on a hyperparameter α (we used $\alpha = 0.5$ because it worked, experimentally and only in some cases, better than least squares loss),

i.e.,

$$Ridge(X_{train}) = \sum_{i=1}^{n} (y_i - f_\beta(x_i)) + \alpha \|\beta\|_2^2 .$$

The features used to train both linear models were:

1. Persistence pooling of 1 element (maximum persistence).
2. Average lives and midlives.
3. Average births and average deaths.
4. Average births and average deaths squared.
5. A combination of births and death squared, average births and deaths, average lives and midlives squared and averages lives and midlives.
6. Persistent entropy.
7. Number of points in persistence diagrams.
8. Linear regression with all the pixels of persistence images.
9. Averages and standard deviations of births and deaths.
10. Averages and standard deviations of births and deaths squared.
11. Vectors of the 10 leading coefficients of the complex polynomial generated from the morphism T (see 2.52).
12. Persistence landscape norms.

These features were computed using own-developed methods and `Giotto TDA` [33] methods package. To compute the input vector to the linear model, we computed each feature on 0, 1 and 2-persistence diagrams and we concatenated the results into a single vector.

For each loss and each different feature we performed **ten** different experiments, averaging their results into a single, more robust than each experiment score individually, R^2 score for each loss and feature. Each experiment consisted of:

1. Shuffling the entire dataset of persistence diagrams and generalization gaps, dividing the dataset in two training and test datasets, consisting of 70% and 30% of the data respectively.
2. Training the model with the chosen loss using the training dataset.
3. Computing the R^2 score of the model using the test dataset.

The R^2 score (**coefficient of determination**) computed for each model is a score ranging from $-\infty$ to 1 that indicates how well the predictions of a model adapt to their real values. The higher the score, the better the predictions are. A score of 1 implies that the predictions fit perfectly their real values. As predictions can be arbitrarily different from the real data, there is no bound to the negative side of the range.

Rigorously, the R^2 score between the predictions $\{\hat{y}_i = f_\beta(x_i)\}_{i\in I}$ of a given model f_β and the real data $\{y_i\}_{i\in I}$ in a dataset $\{(x_i, y_i)\}_{i\in I}$ where $I = \{1, ..., m\}$ is

$$R^2 = 1 - \frac{\sum_{i=1}^{m} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{m} (y_i - \bar{y})^2},$$

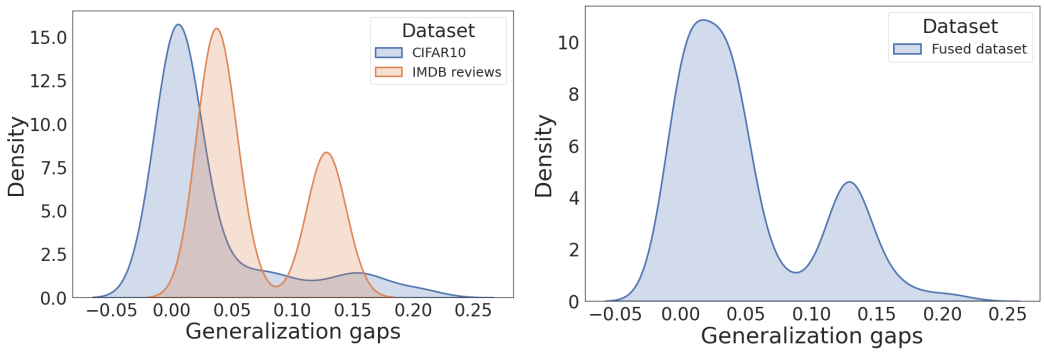
where $\bar{y} = \frac{1}{m} \sum_{i=1}^{m} y_i$.

Chapter 4

Results and visualizations

4.1 Generalization gap distributions

In 4.1, it can be observed how the generalizations gaps are distributed. The first characteristic that can be noticed is that generalization gap distributions differ notably for both datasets. Moreover, the generalization gaps for CIFAR10 and IMDB reviews are not equally distributed in the possible generalization gap ranges. This is of fundamental importance in the analysis, because it means that our linear models were not trained with a representative sample of the generalization gap space in both problems separately. This may mean that linear regressions are not trained to generalize properly in all the generalization gap spectrum, focusing on low generalization gaps in CIFAR10 and in the two clusters that we can observe in the figure 4.1a in IMDB reviews. Hence, adding more examples with uncommon generalization gaps may improve the results given in this section. Moreover, if we look at 4.1b, we see that, although there are two clusters in the fused dataset distribution too, the values are better spread over the spectrum, giving a better dataset to analyze.



(a) IMDB reviews and CIFAR10 datasets.

(b) Fused dataset.

Figure 4.1: Density plots of the IMDB reviews and CIFAR10 generalization gap distributions.

4.2 Visualizations and final results

The heatmap figures 8, 9 and 10 contain the average R^2 scores for each regression modality using the different combinations of dimensions for each feature type. For example, the column *"Classic* 1, 2*"* contains R^2 scores computed using features in the 1 and 2 persistence diagrams (concatenated) in a linear regression with least squares loss whereas *"Ridge* 0*"* column contains R^2 scores using features in the 0-persistence diagram in a linear regression with Ridge linear loss. Also, the bar plots 11, 12 and 13 compare the maximum R^2 scores for each feature across all the possible dimensions and regressions.

As we can observe, in both individual datasets we found features that improve the results given in [10] using the R^2 score. Persistent entropies trained with a Ridge linear regression using persistent dimensions 0, 1 and 2 has higher value than all the values in the column of lives and midlives for the IMDB reviews dataset. Moreover, in the CIFAR10 dataset, averages and standard deviations of births and deaths for dimensions 1 and 2 in a classic linear regression has the highest value of the whole heatmap next to the combination of multiple features. In fact, in the fused model, the maximum R^2 scores for persistent entropies and averages and standard deviations were 0.35 and 0.48, substantially higher than the average lives and midlives R^2 score feature counterpart, that had a maximum R^2 score of 0.25.

Persistence landscape norms gave very poor results in all datasets overall. This means that, in order to take profit of persistence landscapes, we should look for another way of extracting useful information, as taking the norms of the different persistence landscapes seems to discard a lot of information about the whole structure.

Persistence images, though they did not have as good results as other methods, did not perform poorly in each scenario. This makes sense due to the complexity of studying images as a whole. In fact, a linear regression of all their pixels was a very naive approach, only useful to see if we found some overall characteristics in them (in fact, computer vision is a field where many techniques have been developed to deal with image analysis complexity). A more complex analysis should be done before stating if persistence images are useful or not in this particular problem.

Another interesting point to note is that persistence pooling of one element was one of the worst features analyzed. While this can be seen as unimportant, this opens a new research field to analyze if intrinsic properties of neural networks depends on the points of the persistence diagrams or if, instead, they depend on their distribution (variability, average and so on).

We can observe that, in general, quadratic models perform better than their linear counterparts. This could mean that the relationships between generalization gaps and persistent homology are not linear and we should search more complex models to analyze (in 4, we can see that, although there is a clear model for CIFAR10, it is definitely not linear).

4.2.1 Leading coefficients of the complex polynomial generated from the morphism T

We should note that, if we discard the non-linear models, the best model in the fused dataset was the one yielded by the vectors of the 10 leading coefficients of the complex polynomial generated from $T(P)$ (see 2.52). These features, visualized at 5, show a clear correlation with generalization gaps in both models individually and as a whole for dimensions one and two. In fact, the most related component of the feature in our graphs is $Im\left(a_{|P|-1}\right)$, that is the imaginary part of the coefficient $a_{|P|-1}$ in the polynomial $x^{|P|} + \sum_{i=0}^{|P|-1} a_i x^i$.

Actually, if we look at 6, we can see that, although there is a lot of noise, the generalization gap increases whenever $Im\left(a_{|P|-1}\right)$ increases, especially for dimensions one and two, opening the doors to extract a unified model to analyze generalization in neural networks. A projection to this component can be seen in 6.

For dimension zero we see the common behaviour that we can see in other features: dimension zero is the one with less structure in our visualizations (see 1, 2, 3, 4 and 5). Moreover, dimension zero is the one with the worst distributions of features regarding generalization gap. In fact, except for two of the features (of a total of 13 ones) for the IMDB reviews dataset, R^2 scores of dimension zero are lower than the R^2 score of dimension one (and five of thirteen and seven of thirteen for the fused dataset and the CIFAR10 dataset respectively). Dimension zero homology, in particular, is related with the quantity of connected components in the simplicial complex (in our research, the more correlation between neurons, the earlier they get connected in the filtration). Moreover, as we do not take into account the graph's directed edges when connecting neurons in the activation graph, we are neglecting the network's graph structure, allowing any kind of simplex to be built without restrictions. This leads to the following conjecture:

Conjecture 4.1. *Given a computational graph G and its activation graph, equipped with the correlation pseudo-semimetric defined in the problem, i.e., $d_X(x,y) = 1 - |corr_X(x,y)|$, then some intrinsic properties of the given computational graph related with generalization gap are discarded when computing its persistence diagrams.*

This conjecture led us to define as future work adding the network structure to the study. An early approach to do that would be computing **flag complexes** (see [24]) to compute persistence diagrams based only on the graph structure of the network.

Another interesting property (that is only, again, a conjecture), which we can see in our feature visualizations (and it holds in almost every feature, including complex polynomials), is that dimensions one and two almost identical, and therefore they should contribute with the same information to the generalization problem.

Conjecture 4.2. *Given a computational graph G and its activation graph, equipped with the correlation pseudo-semimetric defined in the problem, i.e., $d_X(x,y) = 1 - |corr_X(x,y)|$, then the bottleneck distance between the persistence diagrams extracted from the activation graph for dimensions one and two is small.*

4.2.2 Context dependence in generalization analysis

Averages and standard deviations of births and deaths yielded a very important result that should be analyzed. In the figure 2, we can see an extraordinary well-fitted symmetry between the generalization gap distributions of both datasets according, especially, to standard deviation of births (and deaths) for dimensions one and two. Although this may seem contradictory with the previous subsection, it confirms that some topological features (for example, we can see that a similar symmetry can be found in the figure 3 for dimension zero too) depends strongly on the problem context, and not exclusively in the network structures (like the previous one seemed to do). This relates directly with the fact that it does not exist a neural network that works perfectly after training in a problem dataset and the reason why specific neural networks are developed in each machine learning area, having, each of them, specific properties that improves the performance in particular datasets.

We can extract two conclusions from that:

1. Some topological features are more fine-grained detecting intrinsic network structure properties, like averages and standard deviations of births and deaths.

2. Some topological features are better capturing coarse-grained network structure properties that are common to all kind of machine learning problems and deep learning structures, like Vectors of the 10 leading coefficients of the complex polynomial generated from the morphism T

In the first case, we deduce that some topological features extract more information on the features influenced by the dataset itself, and not only by the network structure. This may explain why we have this symmetry in 2, where we can see clearly that, for the CIFAR10 the best results are obtained when the standard deviation of the birth (and death) points in the persistence diagram and, for the IMDB reviews, the best results are obtained in the opposite case.

In the second case, we deduce that some of the relations between the points in the persistence diagram are strongly related with the formal network structure itself, more than with the problem context. This is very nice because we may discover that there exists networks that, properly adapted, work better overall (context-free). That would be a nice discover because it would allow Machine Learning practitioners to focus only in developing nice networks overall, opening a new field of study of general-purpose networks that could be analyzed using persistent homology.

These approaches are complementary and stablish nice starting points to classify methods in two different branches: general-purpose and context-dependent network generalization analysis.

Chapter 5

Conclusions

We trained several different underfitted, nice fitted and overfitted neural networks using two different datasets, *IMDB reviews* and *CIFAR10*, that belong to two different machine learning problems, object recognition from images and sentiment analysis, to guarantee independence in the input data and thus independence in the features that a neural network should learn to generalize properly.

Once we had these networks trained, we computed their activation graphs in order to obtain their persistence diagrams and then map them to useful homology features to predict distributions of generalization gaps in both problems.

While there is no clear unified linear model that describes properly both generalization gap distributions together, we can observe that one feature stands out from the others, having a strong correlation with generalization gaps overall. This feature is the vector of the 10 leading coefficients of the complex polynomial generated from the morphism $\boldsymbol{T}$ whose visualizations in the figure 5 distinguish clearly that high and low generalization gaps are clustered, following, to a greater or lesser extent, a pattern. Due to that, this feature allows an interpretation for both problems together and it yields an *"impressive value"* (as a baseline) of 0.44 for the fused dataset.

Regarding both datasets individually, the maximum R^2 scores found in both problems are relatively high for a novel approach like this, being 0.48 using standard deviations and averages of births and deaths and an impressive quantity of 0.76 using persistent entropies for the CIFAR10 and the IMDB reviews datasets respectively. What is more, in both problems and in the fused dataset as well, we improved state-of-the-art linear regression given by Corneanu et al. in [10].

The fact that the best model for each problem is different than the best model for the fused dataset and the fact that some topological feature models are symmetric for both problems indicate that there exist topological features that depend on the dataset directly and other ones that are robust to context change. This opens the doors to study specific problem generalization and context-free generalization separately.

In particular, persistent entropies yielded promising expectations about predicting generalization gaps in some particular datasets, because they performed badly in the CIFAR10 dataset but were the best topological features in the IMDB reviews dataset. Moreover, standard deviations and averages of births and deaths had extraordinarily good results

in the CIFAR10 dataset (achieving maximum score in this problem) and they maintained nice results for the IMDB reviews dataset as well (some of the best results we achieved if we discard quadratic models). Furthermore, the latter model yielded interesting results about dataset dependent generalization analysis, making them strong candidates to be related with generalization gaps when fixing the study to only one dataset.

However, these results generate more questions than answers they give, because, although we got interesting new results, many hyperparameters were chosen experimentally and they are open to be changed and reinterpreted, like the metric in the activation graph, the probability and weight functions of persistence images or the complexes used in persistent filtrations. Not only that, but also we analyzed only a very restricted kind of networks (multilayer perceptrons) following a restricted shape (Pareto distribution's shape) in only two different datasets. Due to that, questions about how diversity in models (convolutional neural networks, recurrent models, transformers...) and in problems (text procesing, audio classification...) affects topological features and their connection with generalization gaps arise.

In conclusion, we followed Corneanu's path [11, 10] to study the neural network generalization problem while giving new rigorous statistical features and definitions to let other researchers get their hands on.

5.1 Future work

There is much work that remains to be done. For example, exploring new metrics to construct the activation graph. Better pseudo-semimetrics (or even metrics) than our modification of correlation could be achieved for sure. Some of our ideas with respect to this topic include metrics based on graph properties (without using examples), correlation between Jacobians, weighted averages between properties, etc. This is actually an open problem that could allow us to reinterpret what the structure of a neural networks mean. Not only that but also filtrations should be revisited: using flag complexes to generate filtrations taking into account the direction of the edges in the computational graph could improve the results of this work, restricting the kind of complexes that should be analyzed (and this can mean, on one hand, removing noise, or, on the other hand, discarding important information about the network graph.)

Another research topic is related with persistence modules. Once we have a persistence diagram, which is the most descriptive topological feature? Is there any empirical formula that approximates correlation without training ML models like linear regressions?

A third research area is how to handle thousands (and perhaps millions) of neurons in a neural network. The more complex a network is, the more expensive is the computation of its persistence diagrams. Regarding that, some advances have been done like parallelizing Ripser (**Ripser++**, see [36]).

More practical research areas regard to analyze topological properties on more complex deep neural networks structures, like RNNs, CNNs, Autoencoders... We do not know even if the topological properties of both datasets' networks share the same intrinsic properties about the network.

As you can see, this novel area admits many possibilities for investigation, and we are

citing only a few ways to continue the project, but the fact that TDA is one of the fastest growing areas of research in applied mathematics allows us to go beyond our results, that may be outdated in some months or years from now.

Bibliography

[1] Henry Adams et al. "Persistence Images: A Stable Vector Representation of Persistent Homology". In: *Journal of Machine Learning Research* 18.8 (2017), pp. 1–35. URL: http://jmlr.org/papers/v18/16-337.html.

[2] Nieves Atienza, Rocío González-Díaz, and Manuel Soriano-Trigueros. "On the stability of persistent entropy and new summary functions for topological data analysis". In: *Pattern Recognition* 107 (Nov. 2020), p. 107509. ISSN: 0031-3203. DOI: 10.1016/j.patcog.2020.107509. URL: http://dx.doi.org/10.1016/j.patcog.2020.107509.

[3] Ulrich Bauer. *Ripser: efficient computation of Vietoris-Rips persistence barcodes*. 2019. arXiv: 1908.02518 [math.AT].

[4] Ethan Bolker. *Why does the Betti number give the measure of k-dimensional holes?* URL: https://math.stackexchange.com/q/3206363.

[5] Thomas Bonis et al. "Persistence-based pooling for shape pose recognition". In: *CTIC* (2016), pp. 19–29.

[6] Peter Bubenik. "The Persistence Landscape and Some of Its Properties". In: *Abel Symposia* (2020), pp. 97–117. ISSN: 2197-8549. DOI: 10.1007/978-3-030-43408-3_4. URL: http://dx.doi.org/10.1007/978-3-030-43408-3_4.

[7] Yu-Min Chung and Austin Lawson. *Persistence Curves: A canonical framework for summarizing persistence diagrams*. 2020. arXiv: 1904.07768 [cs.CG].

[8] David Cohen-Steiner, Herbert Edelsbrunner, and John Harer. "Stability of Persistence Diagrams". In: *Proceedings of the Twenty-First Annual Symposium on Computational Geometry*. SCG '05. New York, NY, USA: Association for Computing Machinery, 2005, pp. 263–271. ISBN: 1581139918. DOI: 10.1145/1064092.1064133. URL: https://doi.org/10.1145/1064092.1064133.

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

[10] Ciprian Adrian Corneanu et al. "Computing the Testing Error Without a Testing Set". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 2674–2682. DOI: 10.1109/CVPR42600.2020.00275.

[11] Ciprian Adrian Corneanu et al. "What Does It Mean to Learn in Deep Networks? And, How Does One Detect Adversarial Attacks?" In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 4752–4761. DOI: 10.1109/CVPR.2019.00489.

[12] Tammo tom Dieck. *Algebraic Topology*. EMS Textbooks in Mathematics. European Mathematical Society, 2008. ISBN: 9783037190487.

[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] Barbara Di Fabio and Massimo Ferri. "Comparing persistence diagrams through complex vectors". In: *Proc. of the 6th Nordic Conf. on Human-Computer Interaction*. Vol. 9279. Lecture Notes in Computer Science. Springer, 2015.

[15] Brittany Terese Fasy et al. "Confidence sets for persistence diagrams". In: *The Annals of Statistics* 42.6 (2014), pp. 2301–2339. URL: http://www.jstor.org/stable/43556495.

[16] Chad Giusti et al. "Clique topology reveals intrinsic geometric structure in neural correlations". In: *Proceedings of the National Academy of Sciences* 112.44 (2015), pp. 13455–13460. ISSN: 0027-8424. DOI: 10.1073/pnas.1506407112. eprint: https://www.pnas.org/content/112/44/13455.full.pdf. URL: https://www.pnas.org/content/112/44/13455.

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

[18] He-Liang Huang et al. *Demonstration of Topological Data Analysis on a Quantum Processor*. 2019. arXiv: 1801.06316 [quant-ph].

[19] Yiding Jiang et al. *NeurIPS 2020 Competition: Predicting Generalization in Deep Learning*. 2020. arXiv: 2012.07976 [cs.LG].

[20] John Jumper et al. "High accuracy protein structure prediction using deep learning". 2020.

[21] Alex Krizhevsky. *Learning multiple layers of features from tiny images*. Tech. rep. 2009. URL: https://www.cs.toronto.edu/~kriz/cifar.html.

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

[23] Dimitri Lozeve. "Topological data analysis of temporal networks". MA thesis. University of Oxford, 2018.

[24] Daniel Luetgehetmann et al. *Computing persistent homology of directed flag complexes*. 2019. arXiv: 1906.10458 [math.AT].

[25] Andrew L. Maas et al. "Learning Word Vectors for Sentiment Analysis". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*. HLT '11. Association for Computational Linguistics, 2011, pp. 142–150. URL: http://dl.acm.org/citation.cfm?id=2002472.2002491.

[26] Marc Mezard and Andrea Montanari. *Information, Physics, and Computation*. Oxford University Press, 2009. ISBN: 019857083X.

[27] *numpy.corrcoef*. Accessed: 2020-01-22. URL: https://numpy.org/doc/stable/reference/generated/numpy.corrcoef.html.

[28] Nina Otter et al. "A roadmap for the computation of persistent homology". In: *EPJ Data Science* 6.1 (Aug. 2017). ISSN: 2193-1127. DOI: 10.1140/epjds/s13688-017-0109-5. URL: https://doi.org/10.1140/epjds/s13688-017-0109-5.

[29] *Ripser.py 0.6.0 documentation - Greedy permutation*. https://ripser.scikit-tda.org/en/latest/notebooks/Greedy%20Subsampling%20for%20Fast%20Approximate%20Computation.html. Accessed: 2020-01-22. 2007.

[30] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. *Deep inside convolutional networks: visualising image classification models and saliency Maps*. 2014. arXiv: 1312.6034 [cs.CV].

[31] *sklearn.metrics.r2_score*. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html. Accessed: 2020-01-22.

[32] Elchanan Solomon, Alexander Wagner, and Paul Bendich. *A Fast and Robust Method for Global Topological Functional Optimization*. 2020. arXiv: 2009.08496 [stat.ML].

[33] Guillaume Tauzin et al. *giotto-tda: A Topological Data Analysis Toolkit for Machine Learning and Data Exploration*. 2020. arXiv: 2004.02551 [cs.LG].

[34] Tim. *Correlation with a constant*. URL: https://stats.stackexchange.com/q/267155.

[35] Matthew D Zeiler and Rob Fergus. *Visualizing and Understanding Convolutional Networks*. 2013. arXiv: 1311.2901 [cs.CV].

[36] Simon Zhang, Mengbai Xiao, and Hao Wang. *GPU-Accelerated Computation of Vietoris-Rips Persistence Barcodes*. 2020. arXiv: 2003.07989 [cs.CG].

Appendices

A.1 Objectives and planification

Our objectives were the following ones:

1. Develop a simple enough dataset of neural networks to analyze some of their topological features.
2. Develop a software that, given an arbitrary Tensorflow model, returns its persistence diagrams using the activation graph defined in [11].
3. Develop an intuitive model that predicts the generalization gap of a network given its persistence diagrams computed in the previous step (and improving the state of the art).
4. (Bonus) Participate in the NeurIPS generalization challenge.

The schedule (and deadlines) of our work is summarized in the next table.

	Objective 1	Objective 2	Objective 3	Objective 4
July		X		X
August		X	X	X
September		X	X	X
October		X	X	X
November	X			
December	X		X	
January			X	

Note: The 4^{th} objective was impossible to afford due to the complexity of computing persistence diagrams for very deep neural networks (explained in the chapter 3).

A.2 Figures from results

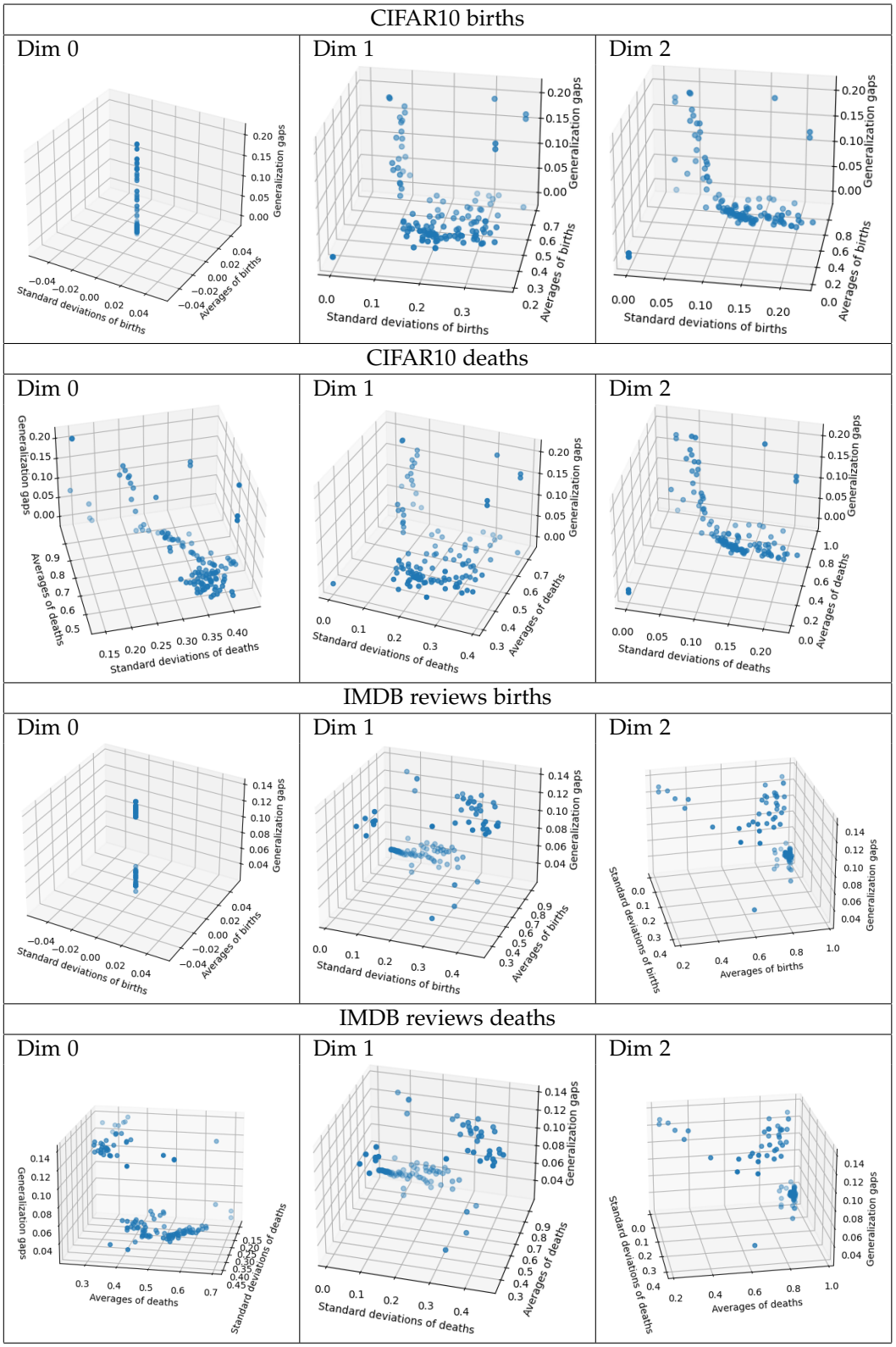


Figure 1: Standard deviations and averages (in the x-axis and y-axis respectively) of births and deaths compared with generalization gaps for CIFAR10 and IMDB reviews datasets. We can observe that, while there is a clear relation between CIFAR10 features and their generalizations gaps, it is not clear that we can rely on the same model for the IMDB reviews dataset.

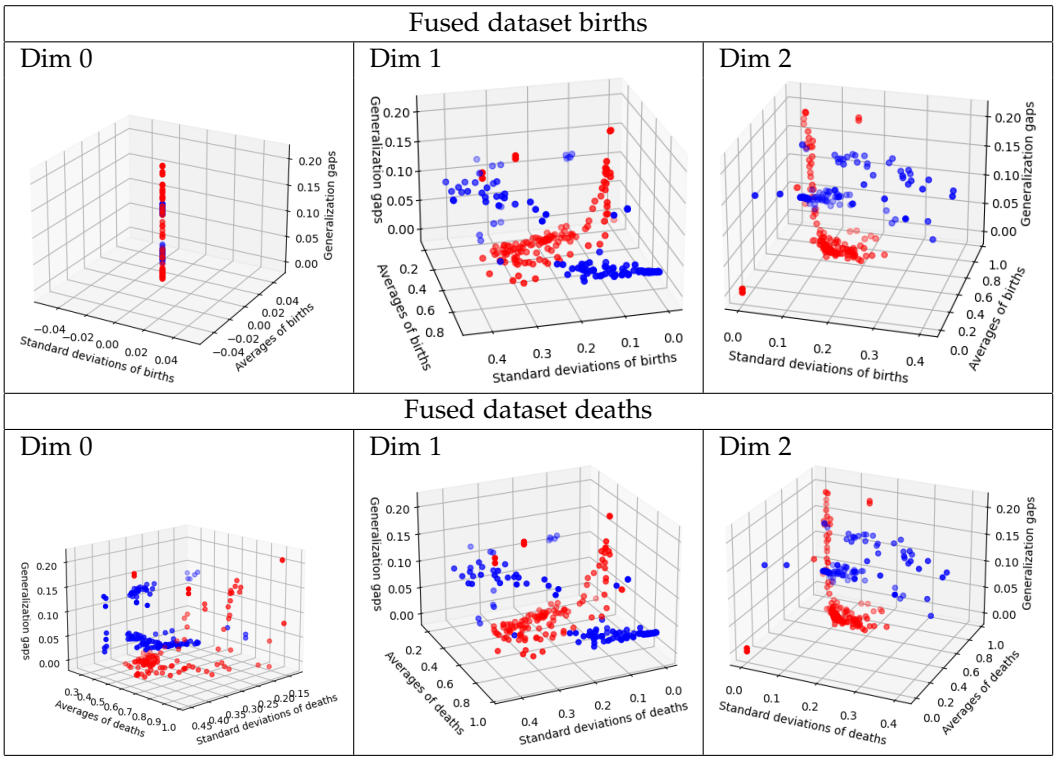


Figure 2: Standard deviations and averages (in the x-axis and y-axis respectively) of births and deaths compared with generalization gaps for the fused dataset (red and blue points for the CIFAR10 and for the IMDB reviews datasets respectively). In the visualizations, we can see that this feature's relationship with generalization gaps changes dramatically between problem datasets. This is especially visible for the dimension one of the feature computed with deaths, where we see a clear symmetry between blue and red points over a perpendicular plane to the standard deviation axis.

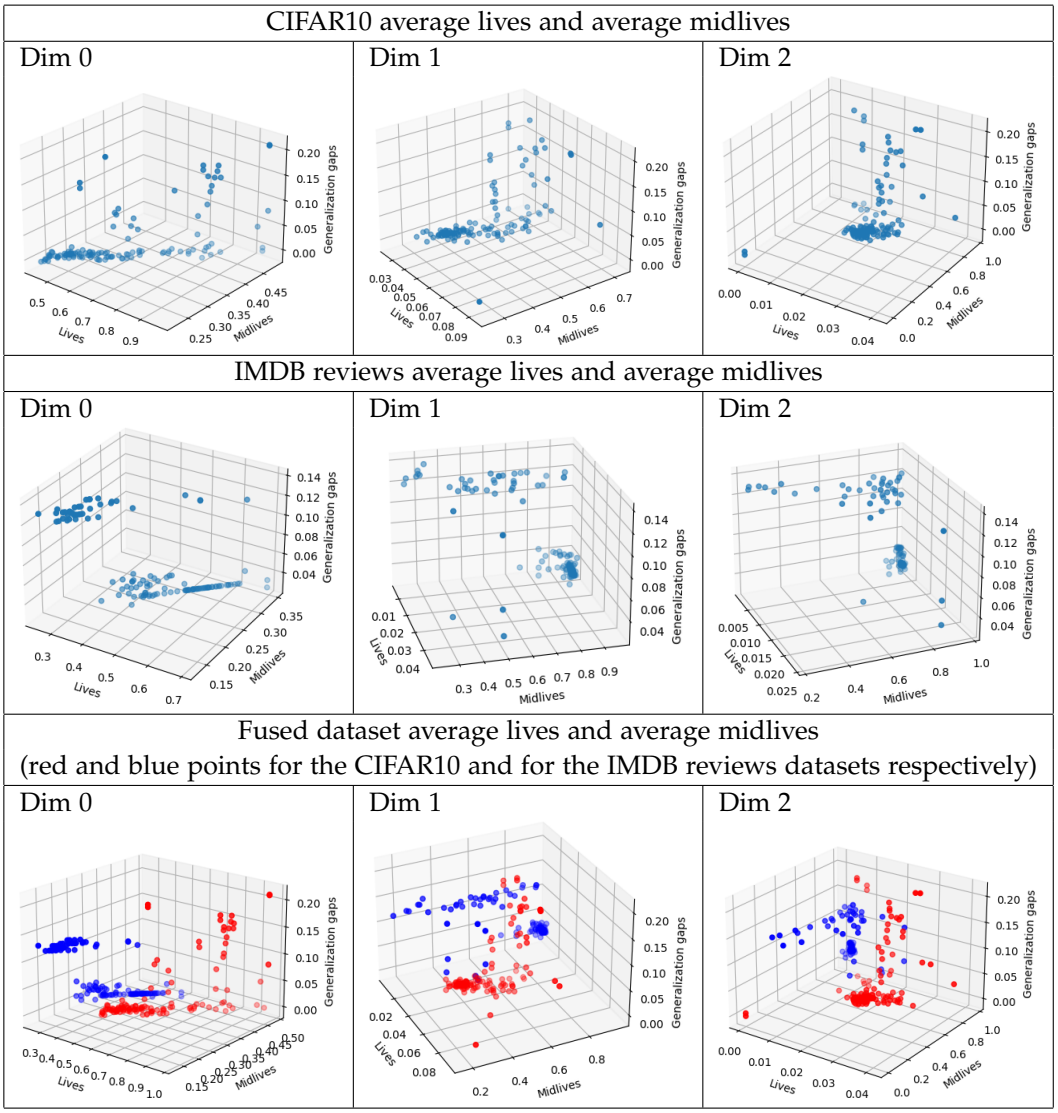


Figure 3: While average lives and average midlives give some of the best results in our experiments, we cannot say that they can be modelled to predict generalization gaps clearly because of the quantity of noisy points that the graph has. Moreover, in the fused model, we can see that this feature seems to be context-dependent as the point cloud does not seem to follow any known distribution for both problems together.

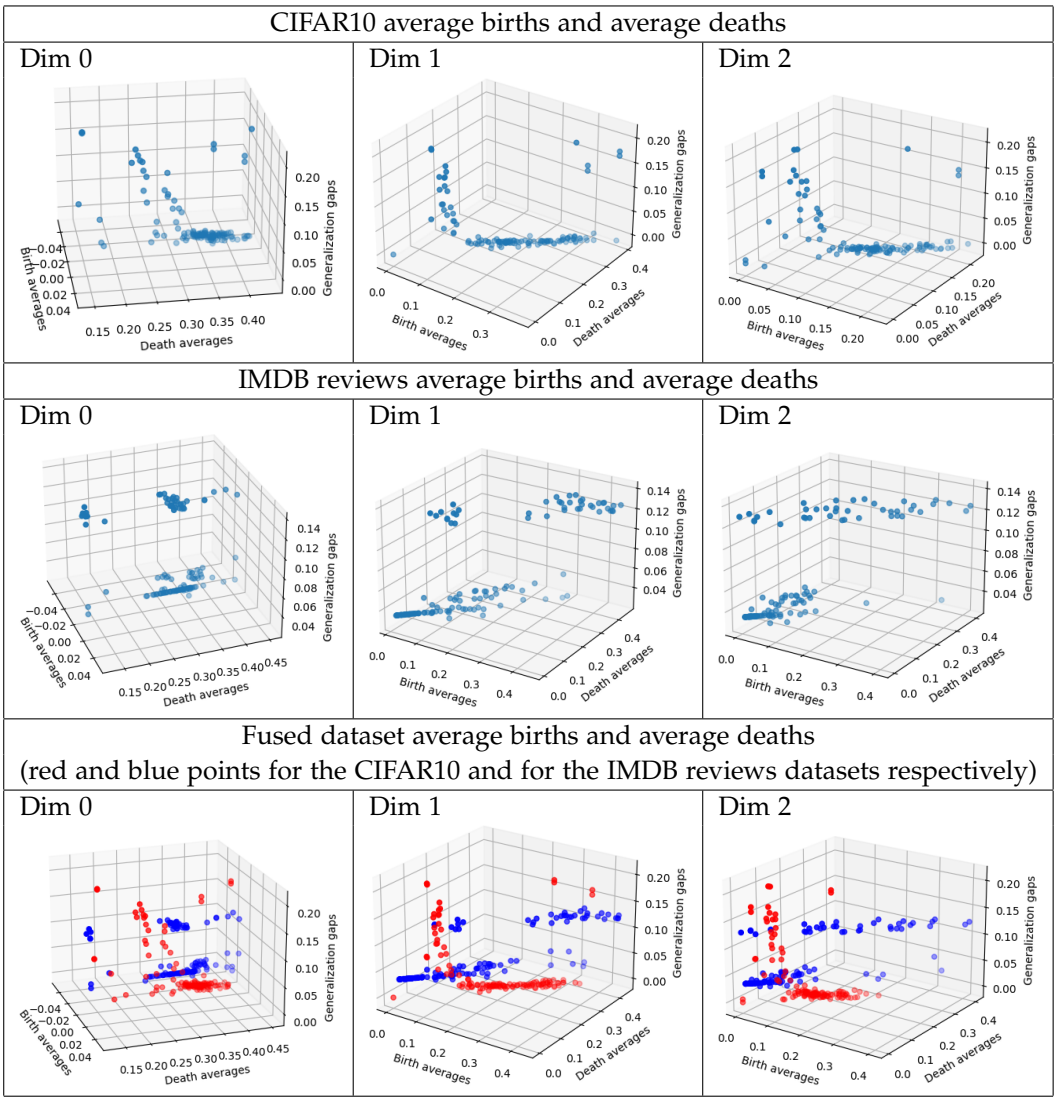


Figure 4: Average births and average deaths give promising visualizations where, in the case of the CIFAR10 dataset, we find a clear relation between generalization gap and these features. However, in the IMDB reviews dataset, the results are not as clear as previously, although, except in some noisy clusters of points, the features divide clearly high and low generalization gaps. The fused model visualizations show that there is no a relation between features and generalization gaps of both problems.

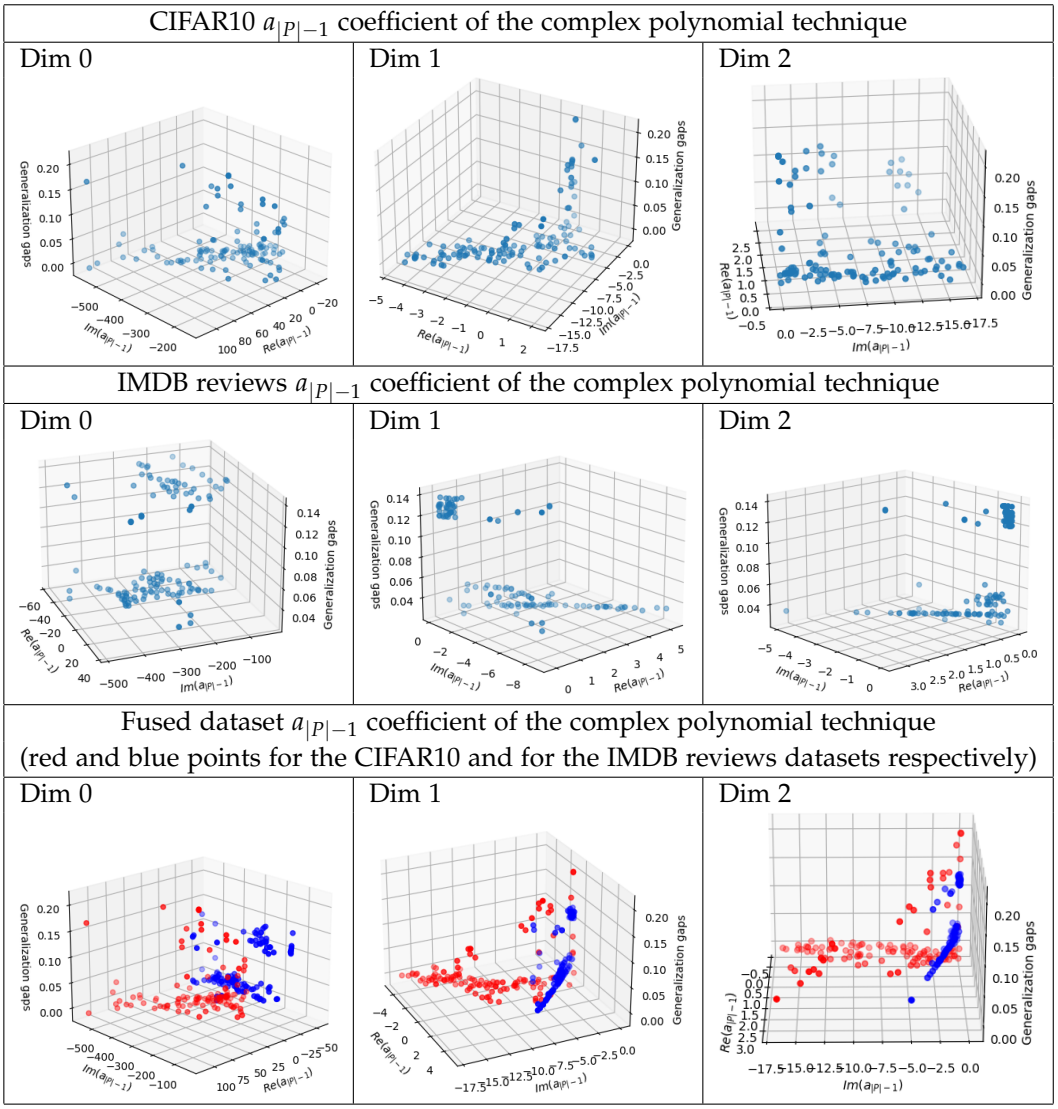


Figure 5: In the image we can visualize the real and imaginary parts of the value $a_{|P|-1}$, $Re\left(a_{|P|-1}\right)$, $Im\left(a_{|P|-1}\right)$ respectively, in the polynomial $x^{|P|} + \sum_{i=0}^{|P|-1} a_i x^i$ generated using the technique of vectors of the 1 latest coefficient of the complex polynomial generated from $T(P)$. These results are strange, because, although they do not perform as nice as other features, it is true that there seems to exist a stable correlation between these coefficients and generalization gaps in both datasets, CIFAR10 and IMDB reviews. Interestingly, in the fused model we can see that the projection to the imaginary part ($Im(|P|-1)$) of the feature seems to follow a pattern (see 6), being the only feature that allows a *possible* interpretation of the fused model.

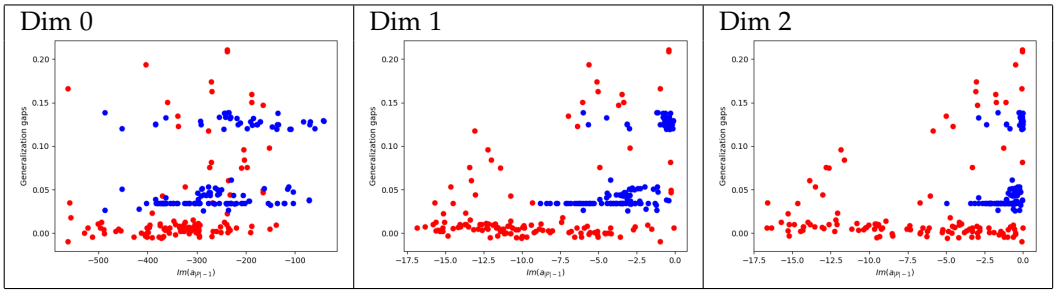


Figure 6: Fused dataset $Im(a_{|P|-1})$ coefficients of the complex polynomial technique.

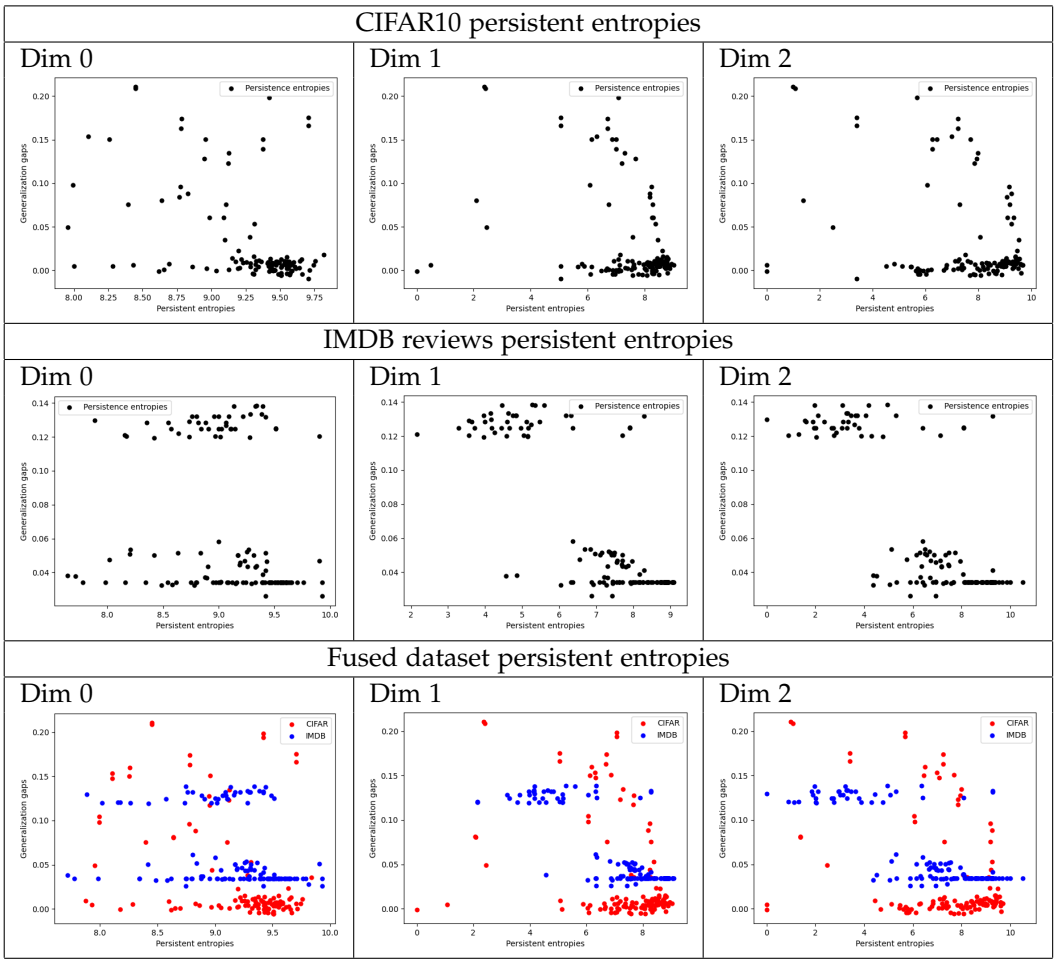


Figure 7: Persistent entropies show a nice behaviour detecting high and low generalization gaps in the IMDB reviews dataset. However, a lot of noise is found when we analyze persistent entropies in the CIFAR10 dataset. For both datasets together, persistent entropies cannot be interpreted easily, although it seems to yield some point clustering patterns when comparing with generalization gaps.

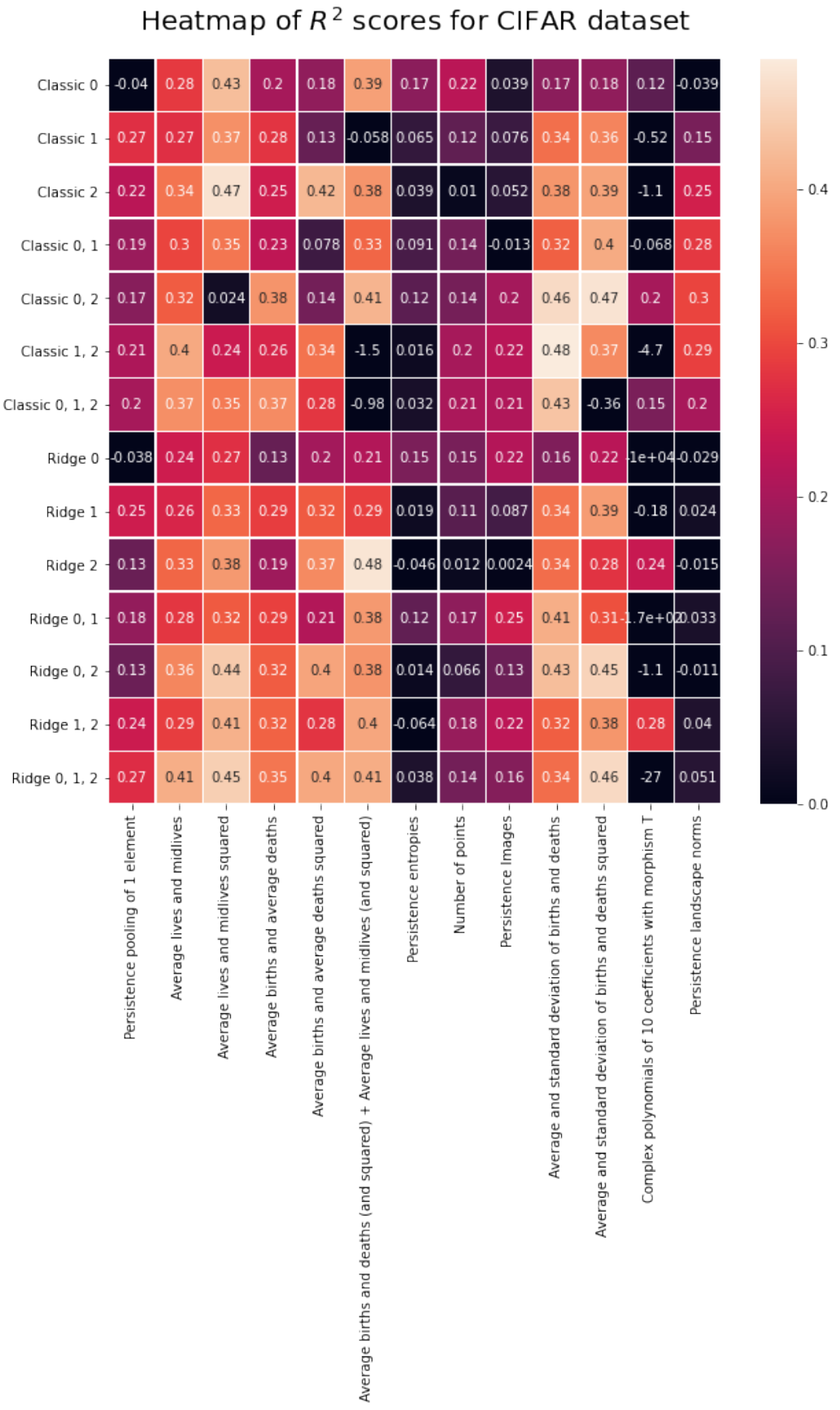

Heatmap of R^2 scores for CIFAR dataset
Classic 0 -0.04 0.28 0.43 0.2 0.18 0.39 0.17 0.22 0.039 0.17 0.18 0.12 -0.039
Classic 1 0.27 0.27 0.37 0.28 0.13 -0.058 0.065 0.12 0.076 0.34 0.36 -0.52 0.15
Classic 2 0.22 0.34 0.47 0.25 0.42 0.38 0.039 0.01 0.052 0.38 0.39 -1.1 0.25
Classic 0, 1 0.19 0.3 0.35 0.23 0.078 0.33 0.091 0.14 -0.013 0.32 0.4 -0.068 0.28
Classic 0, 2 0.17 0.32 0.024 0.38 0.14 0.41 0.12 0.14 0.2 0.46 0.47 0.2 0.3
Classic 1, 2 0.21 0.4 0.24 0.26 0.34 -1.5 0.016 0.2 0.22 0.48 0.37 -4.7 0.29
Classic 0, 1, 2 0.2 0.37 0.35 0.37 0.28 -0.98 0.032 0.21 0.21 0.43 -0.36 0.15 0.2
Ridge 0 -0.038 0.24 0.27 0.13 0.2 0.21 0.15 0.15 0.22 0.16 0.22 -1e+04 -0.029
Ridge 1 0.25 0.26 0.33 0.29 0.32 0.29 0.019 0.11 0.087 0.34 0.39 -0.18 0.024
Ridge 2 0.13 0.33 0.38 0.19 0.37 0.48 -0.046 0.012 0.0024 0.34 0.28 0.24 -0.015
Ridge 0, 1 0.18 0.28 0.32 0.29 0.21 0.38 0.12 0.17 0.25 0.41 0.31 -1.7e+02 0.033
Ridge 0, 2 0.13 0.36 0.44 0.32 0.4 0.38 0.014 0.066 0.13 0.43 0.45 -1.1 -0.011
Ridge 1, 2 0.24 0.29 0.41 0.32 0.28 0.4 -0.064 0.18 0.22 0.32 0.38 0.28 0.04
Ridge 0, 1, 2 0.27 0.41 0.45 0.35 0.4 0.41 0.038 0.14 0.16 0.34 0.46 -27 0.051
Persistence pooling of 1 element
Average lives and midlives
Average lives and midlives squared
Average births and average deaths
Average births and average deaths squared
Average births and deaths (and squared) + Average lives and midlives (and squared)
Persistence entropies
Number of points
Persistence Images
Average and standard deviation of births and deaths
Average and standard deviation of births and deaths squared
Complex polynomials of 10 coefficients with morphism T
Persistence landscape norms
0.4
0.3
0.2
0.1
0.0

Figure 8

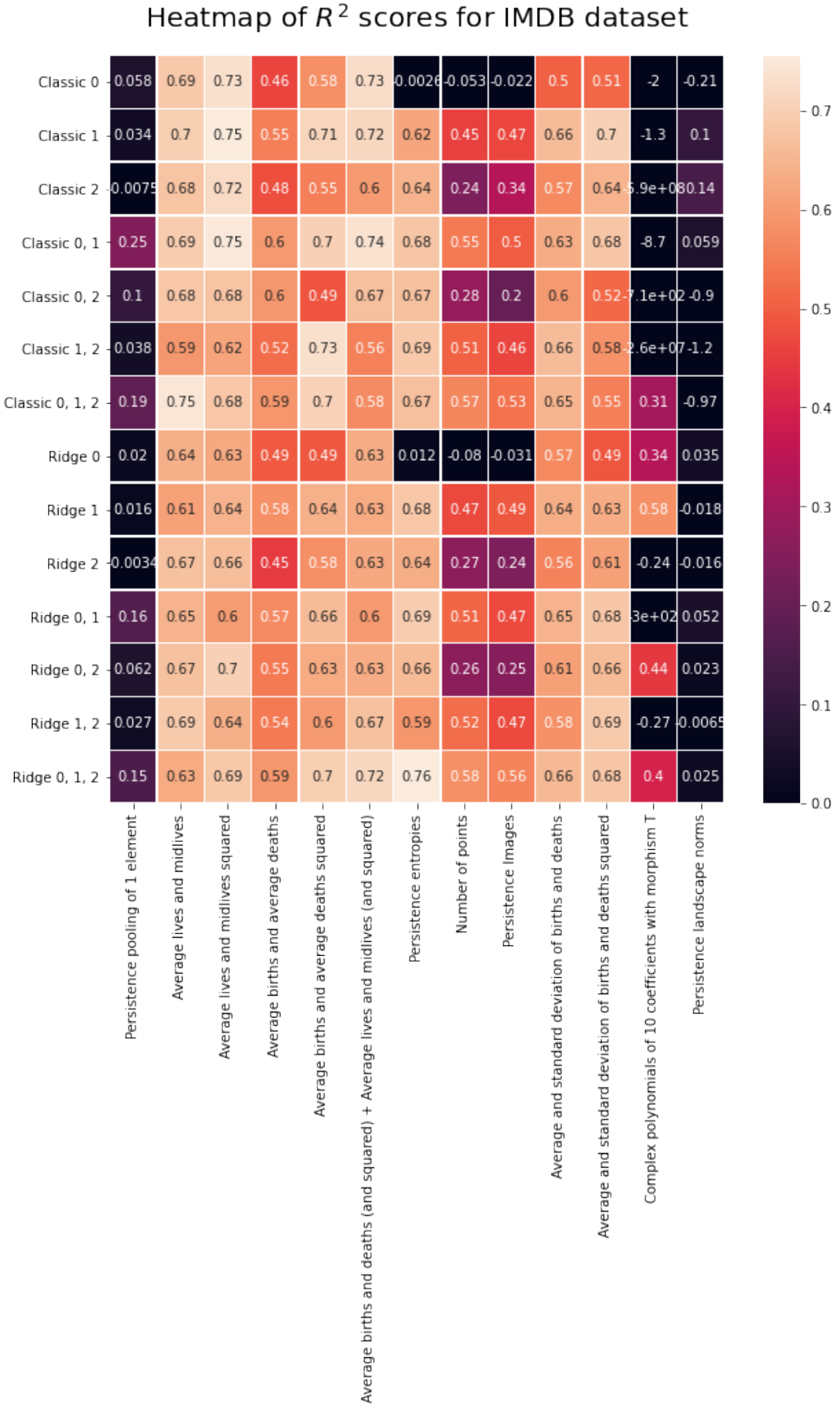

Heatmap of R^2 scores for IMDB dataset
Classic 0 0.058 0.69 0.73 0.46 0.58 0.73 -0.0026 -0.053 -0.022 0.5 0.51 -2 -0.21
Classic 1 0.034 0.7 0.75 0.55 0.71 0.72 0.62 0.45 0.47 0.66 0.7 -1.3 0.1
Classic 2 -0.0075 0.68 0.72 0.48 0.55 0.6 0.64 0.24 0.34 0.57 0.64 -5.9e+08 0.14
Classic 0, 1 0.25 0.69 0.75 0.6 0.7 0.74 0.68 0.55 0.5 0.63 0.68 -8.7 0.059
Classic 0, 2 0.1 0.68 0.68 0.6 0.49 0.67 0.67 0.28 0.2 0.6 0.52 -7.1e+02 -0.9
Classic 1, 2 0.038 0.59 0.62 0.52 0.73 0.56 0.69 0.51 0.46 0.66 0.58 -2.6e+07 -1.2
Classic 0, 1, 2 0.19 0.75 0.68 0.59 0.7 0.58 0.67 0.57 0.53 0.65 0.55 0.31 -0.97
Ridge 0 0.02 0.64 0.63 0.49 0.49 0.63 0.012 -0.08 -0.031 0.57 0.49 0.34 0.035
Ridge 1 0.016 0.61 0.64 0.58 0.64 0.63 0.68 0.47 0.49 0.64 0.63 0.58 -0.018
Ridge 2 -0.0034 0.67 0.66 0.45 0.58 0.63 0.64 0.27 0.24 0.56 0.61 -0.24 -0.016
Ridge 0, 1 0.16 0.65 0.6 0.57 0.66 0.6 0.69 0.51 0.47 0.65 0.68 -3e+02 0.052
Ridge 0, 2 0.062 0.67 0.7 0.55 0.63 0.63 0.66 0.26 0.25 0.61 0.66 0.44 0.023
Ridge 1, 2 0.027 0.69 0.64 0.54 0.6 0.67 0.59 0.52 0.47 0.58 0.69 -0.27 -0.0065
Ridge 0, 1, 2 0.15 0.63 0.69 0.59 0.7 0.72 0.76 0.58 0.56 0.66 0.68 0.4 0.025
Persistence pooling of 1 element
Average lives and midlives
Average lives and midlives squared
Average births and average deaths
Average births and average deaths squared
Average births and deaths (and squared) + Average lives and midlives (and squared)
Persistence entropies
Number of points
Persistence Images
Average and standard deviation of births and deaths
Average and standard deviation of births and deaths squared
Complex polynomials of 10 coefficients with morphism T
Persistence landscape norms
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0

Figure 9

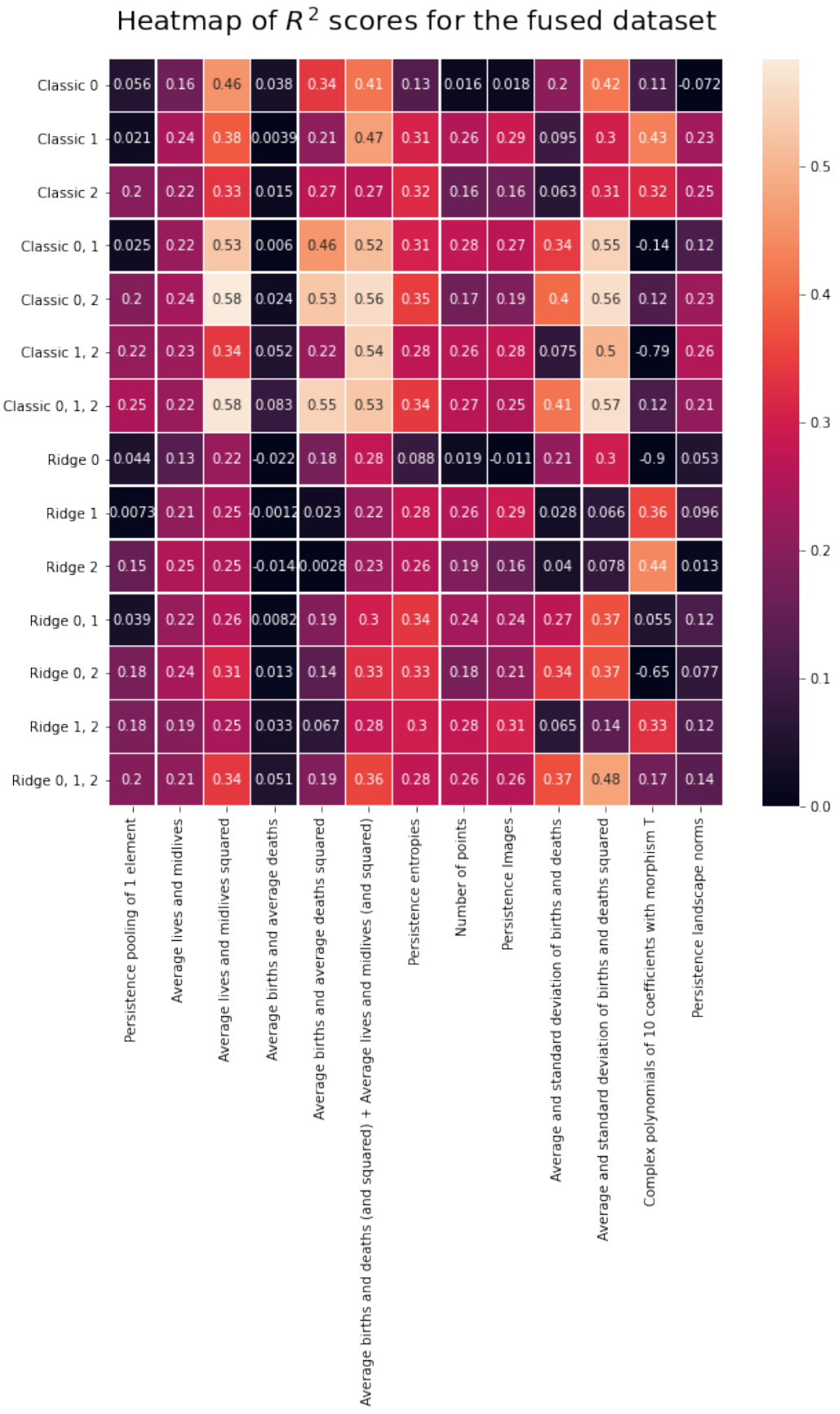

Heatmap of R^2 scores for the fused dataset
Classic 0 0.056 0.16 0.46 0.038 0.34 0.41 0.13 0.016 0.018 0.2 0.42 0.11 -0.072
Classic 1 0.021 0.24 0.38 0.0039 0.21 0.47 0.31 0.26 0.29 0.095 0.3 0.43 0.23
Classic 2 0.2 0.22 0.33 0.015 0.27 0.27 0.32 0.16 0.16 0.063 0.31 0.32 0.25
Classic 0, 1 0.025 0.22 0.53 0.006 0.46 0.52 0.31 0.28 0.27 0.34 0.55 -0.14 0.12
Classic 0, 2 0.2 0.24 0.58 0.024 0.53 0.56 0.35 0.17 0.19 0.4 0.56 0.12 0.23
Classic 1, 2 0.22 0.23 0.34 0.052 0.22 0.54 0.28 0.26 0.28 0.075 0.5 -0.79 0.26
Classic 0, 1, 2 0.25 0.22 0.58 0.083 0.55 0.53 0.34 0.27 0.25 0.41 0.57 0.12 0.21
Ridge 0 0.044 0.13 0.22 -0.022 0.18 0.28 0.088 0.019 -0.011 0.21 0.3 -0.9 0.053
Ridge 1 -0.0073 0.21 0.25 -0.0012 0.023 0.22 0.28 0.26 0.29 0.028 0.066 0.36 0.096
Ridge 2 0.15 0.25 0.25 -0.014 0.0028 0.23 0.26 0.19 0.16 0.04 0.078 0.44 0.013
Ridge 0, 1 0.039 0.22 0.26 0.0082 0.19 0.3 0.34 0.24 0.24 0.27 0.37 0.055 0.12
Ridge 0, 2 0.18 0.24 0.31 0.013 0.14 0.33 0.33 0.18 0.21 0.34 0.37 -0.65 0.077
Ridge 1, 2 0.18 0.19 0.25 0.033 0.067 0.28 0.3 0.28 0.31 0.065 0.14 0.33 0.12
Ridge 0, 1, 2 0.2 0.21 0.34 0.051 0.19 0.36 0.28 0.26 0.26 0.37 0.48 0.17 0.14
Persistence pooling of 1 element
Average lives and midlives
Average lives and midlives squared
Average births and average deaths
Average births and average deaths squared
Average births and deaths (and squared) + Average lives and midlives (and squared)
Persistence entropies
Number of points
Persistence Images
Average and standard deviation of births and deaths
Average and standard deviation of births and deaths squared
Complex polynomials of 10 coefficients with morphism T
Persistence landscape norms
0.5
0.4
0.3
0.2
0.1
0.0

Figure 10

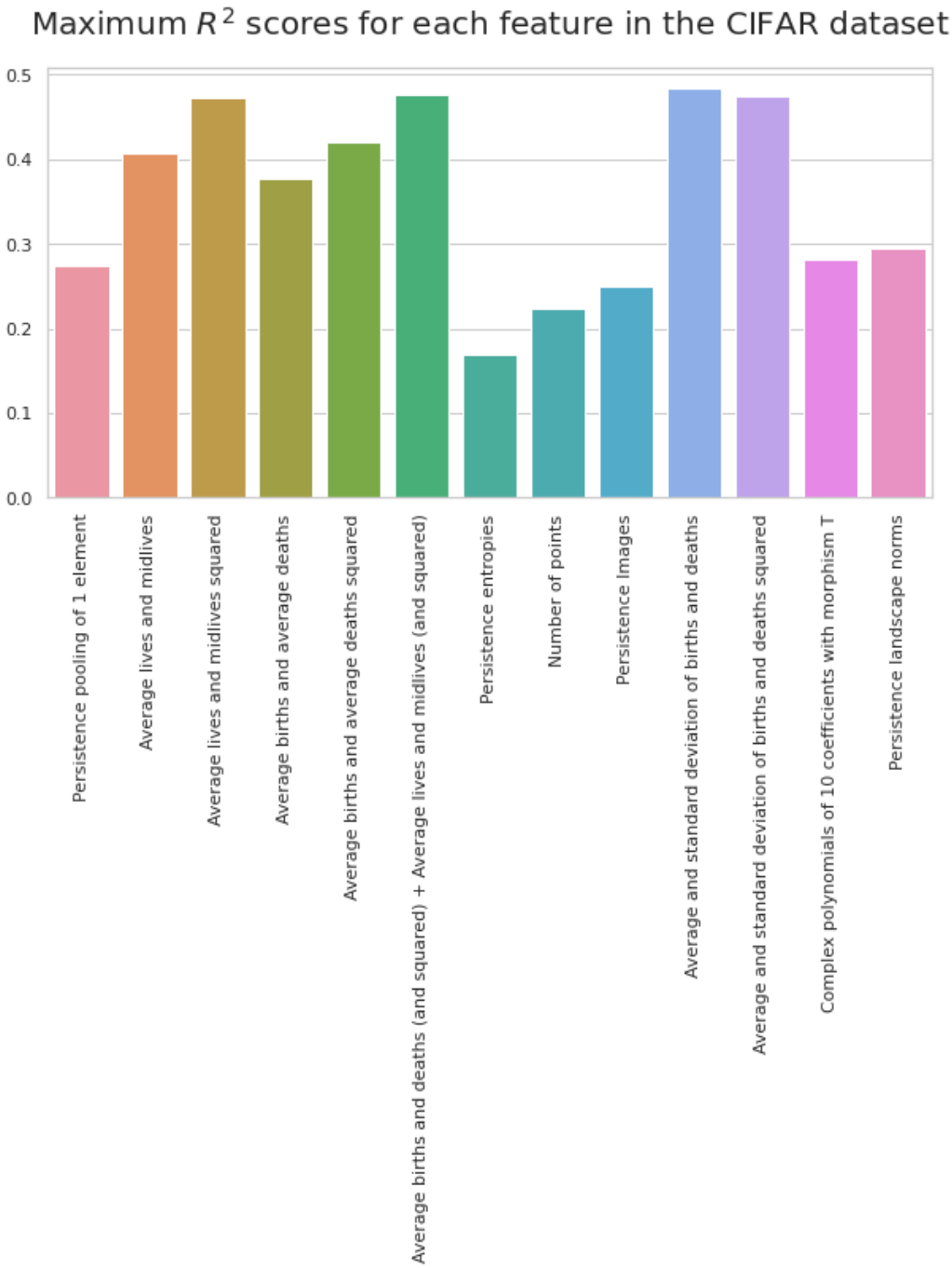

Maximum R^2 scores for each feature in the CIFAR dataset
0.5
0.4
0.3
0.2
0.1
0.0
Persistence pooling of 1 element
Average lives and midlives
Average lives and midlives squared
Average births and average deaths
Average births and average deaths squared
Average births and deaths (and squared) + Average lives and midlives (and squared)
Persistence entropies
Number of points
Persistence Images
Average and standard deviation of births and deaths
Average and standard deviation of births and deaths squared
Complex polynomials of 10 coefficients with morphism T
Persistence landscape norms

Figure 11

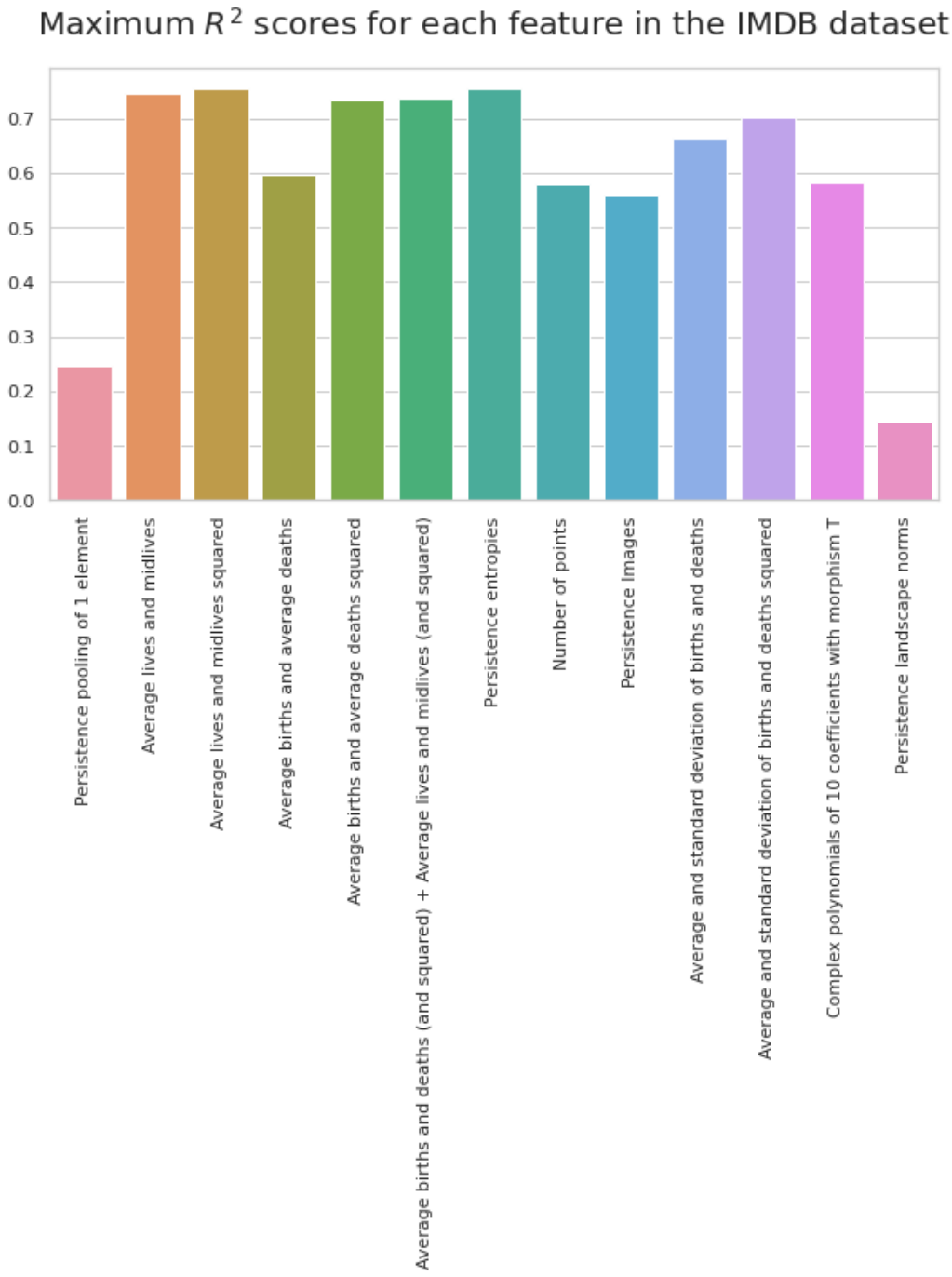

Maximum R^2 scores for each feature in the IMDB dataset
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
Persistence pooling of 1 element
Average lives and midlives
Average lives and midlives squared
Average births and average deaths
Average births and average deaths squared
Average births and deaths (and squared) + Average lives and midlives (and squared)
Persistence entropies
Number of points
Persistence Images
Average and standard deviation of births and deaths
Average and standard deviation of births and deaths squared
Complex polynomials of 10 coefficients with morphism T
Persistence landscape norms

Figure 12

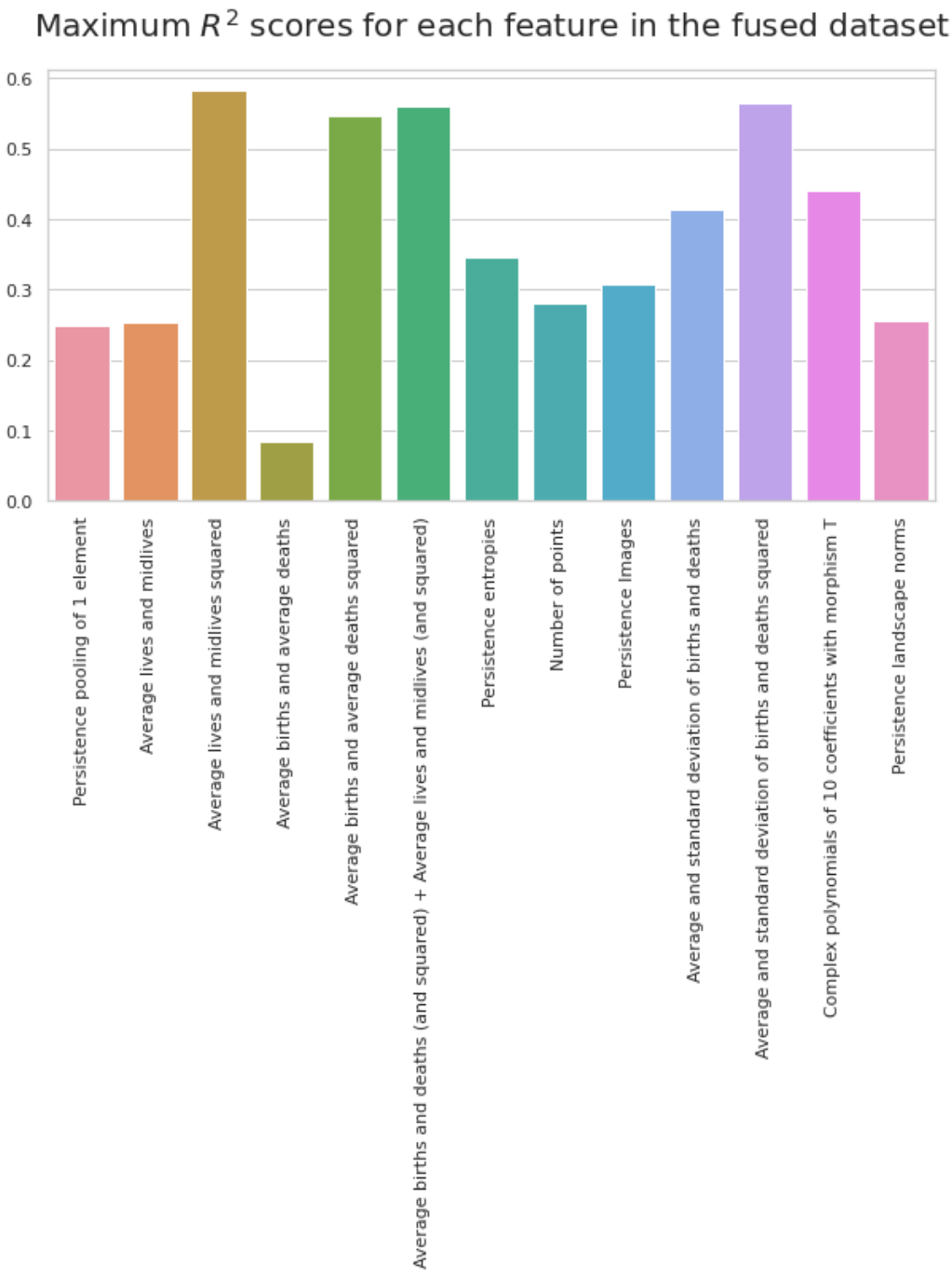

Maximum R^2 scores for each feature in the fused dataset
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Persistence pooling of 1 element
Average lives and midlives
Average lives and midlives squared
Average births and average deaths
Average births and average deaths squared
Average births and deaths (and squared) + Average lives and midlives (and squared)
Persistence entropies
Number of points
Persistence Images
Average and standard deviation of births and deaths
Average and standard deviation of births and deaths squared
Complex polynomials of 10 coefficients with morphism T
Persistence landscape norms

Figure 13