A General Neural Network Architecture for Persistence Diagrams and Graph Classification

Mathieu Carriere, Frédéric Chazal, Yuichi Ike, Théo Lacombe, Martin Royer, Yuhei Umeda

A General Neural Network Architecture for Persistence Diagrams and Graph Classification

Mathieu Carrière
Rabdan Lab
Columbia University
New York, US.
mc4660@columbia.edu

Frédéric Chazal
Datashape, Inria Saclay
Palaiseau, France.
frederic.chazal@inria.fr

Yuichi Ike
Fujitsu Laboratories, AI Lab
Tokyo, Japan.
ike.yuichi@fujitsu.com

Théo Lacombe
Datashape, Inria Saclay
Palaiseau, France.
theo.lacombe@inria.fr

Martin Royer
Datashape, Inria Saclay
Palaiseau, France.
martin.royer@inria.fr

Yuhei Umeda
Fujitsu Laboratories, AI Lab
Tokyo, Japan.
umeda.yuhei@fujitsu.com

Abstract

Graph classification is a difficult problem that has drawn a lot of attention from the machine learning community over the past few years. This is mainly due to the fact that, contrarily to Euclidean vectors, the inherent complexity of graph structures can be quite hard to encode and handle for traditional classifiers. Even though kernels have been proposed in the literature, the increase in the dataset sizes has greatly limited the use of kernel methods since computation and storage of kernel matrices has become impracticable. In this article, we propose to use *extended persistence diagrams* to efficiently encode graph structure. More precisely, we show that using the so-called *heat kernel signatures* for the computation of these extended persistence diagrams allows one to quickly and efficiently summarize the graph structure. Then, we build on the recent development of neural networks for point clouds to define an architecture for (extended) persistence diagrams which is modular and easy-to-use. Finally, we demonstrate the usefulness of our approach by validating our architecture on several graph datasets, on which the obtained results are comparable to the state-of-the-art for graph classification.

1 Introduction

Many different areas of science require working with graph data, such as molecular biology, finance and social sciences. Being able to solve machine learning problems with input data given as graphs has thus become a problem of primary importance in the context of big data. Unfortunately, this turns out to be a difficult question, since the space of graphs is not well-structured (in particular it does not have an Euclidean structure): different graphs may have different numbers of vertices and edges with no clear correspondences between them, and many standard operations, such as computing the sum or the mean of a sample of graphs, are not defined in the space of graphs.

To handle this issue, a lot of attention has been devoted to the construction of *kernels* that can handle graphs, since kernel methods are a very handy and common set of tools that can cope with non-structured data. These kernels are constructed with various graph techniques, such as random walks or graphlet isomorphisms. See [29] and the references therein for a complete description of graph kernels presented in the literature. The main issue faced by the use of kernels is scalability: kernel methods usually require the storage of the entire corresponding kernel matrices, which are quadratic with respect to the number of inputs. This becomes quickly prohibitive as the dataset size

Preprint. Under review.

increases. Hence, a more recent line of work has focused on extending *neural networks* for graph data. These networks extend the traditional convolution occuring in convolutional neural networks with various neighborhood aggregation schemes for graph vertices, often leading to scalable architectures whose results are competitive with kernel methods. Again, we refer the interested reader to [27] and the references therein for an overview of the literature.

A third line of work uses *computational topology* to efficiently encode and summarize graph structures in compact descriptors, that can then be processed with the aforementioned techniques. These descriptors are called *persistence diagrams*, which are sets of points in the Eulidean plane $\mathbb{R}^2$. They aim at computing and quantifying the presence and importance of *topological features* in graphs (and eventually higher dimensional combinatorial objects—known as simplicial complexes), such as branches, loops and connected components. This is the approach advocated in two recent works, where persistence diagrams are first computed on the graphs, and then processed by either a kernel [24] or a neural network [15]. Both of these approaches convolve the persistence diagrams with 2D Gaussian functions in order to define the kernel or the first layer of the neural network.

In this article, we follow this third line of work and focus on the interactions between deep learning and computational topology, by studying neural networks that are tailored to handle and process persistence diagrams, and by providing an illustration on graph classification.

Contributions. In this article, our contributions are two fold:

- We provide an efficient and theoretically sound way to encode and summarize graph structures using an extension of ordinary persistence called *extended persistence* and a specific family of functions called *heat kernel signatures*.
- We then provide a very simple (namely, two layers), versatile and competitive neural network architecture to process (extended) persistence diagrams. It is based on the recently proposed DeepSet architecture [28], and encompasses most of the vectorizing approaches for persistence diagrams of the literature.

Note that our architecture is not restricted to graph classification, and can be used as soon as persistence diagrams are given as input, whatever data they were computed from. Moreover, we will soon release publicly the Python code implementing our architecture and persistence diagram computations.

Outline. Section 2 recalls the basics of (extended) persistence theory. Since (extended) persistence diagrams are defined from functions on graph vertices, we present in Section 3 the family of functions that we use to generate our diagrams, the so-called *heat kernel signatures*. Then, we detail our general neural network architecture in Section 4, and we finally confirm the efficiency of our approach with a set of experiments in Section 5.

Notations. In this paper, a graph is denoted by $G = (V, E)$ where V is the set of vertices and E is the set of non-oriented edges, i.e. a subset of $V \times V$ quotiented by the equivalence relation that identifies (v_1, v_2) with (v_2, v_1) for any $v_1 \neq v_2 \in V$. The weight function on edges is denoted by $w_G : E \to \mathbb{R}_+$. A graph is unweighted if all of its edges have same weight 1.

2 Extended persistence diagrams

In this section, we explain how one can encode graph structures with the so-called extended persistence diagram descriptors. We will merely recall the basics of (extended) persistence theory in the context of graphs, and we refer the interested reader to [13, 20] for a thorough description of general (extended) persistence theory for topological spaces, which was originally defined in [12].

Ordinary persistence. Persistence aims at encoding the structure of a pair (G, f), where $G = (V, E)$ is a graph and $f : V \to \mathbb{R}$ is a function defined on its vertices. This is achieved by looking at the *sublevel graphs* $G_\alpha = (V_\alpha, E_\alpha)$ where $\alpha \in \mathbb{R}$, $V_\alpha = \{v \in V : f(v) \leq \alpha\}$ and $E_\alpha = \{(v_1, v_2) \in E : v_1, v_2 \in V_\alpha\}$—see Figure 1 for an illustration. Indeed, making α increase from $-\infty$ to $+\infty$ gives a sequence of increasing subgraphs, i.e. subgraphs that are nested with respect to the inclusion. This sequence starts with the empty graph and ends with the full graph G.

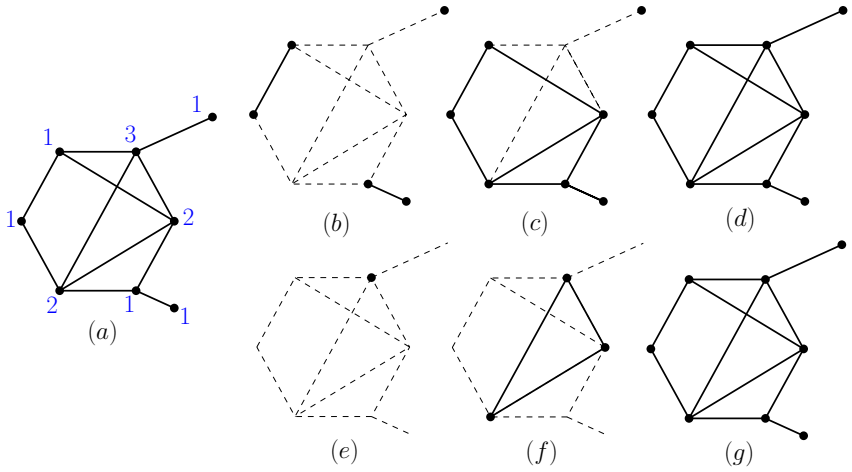

Figure 1: Illustration of sublevel and superlevel graphs. (a) Input graph (V, E) along with the values of a function $f : V \to \mathbb{R}$ (blue). (b) Sublevel graph for $\alpha = 1$. (c) Sublevel graph for $\alpha = 2$. (d) Sublevel graph for $\alpha = 3$, where we recover the input graph. (e, f, g) Superlevel graphs for $\alpha = 3$, $\alpha = 2$ and $\alpha = 1$ respectively.

Persistence aims at encoding the topological changes in the sublevel graphs in this sequence. For instance, any branch of G pointing downwards (with respect to the orientation given by f) will create a new connected component in the sequence of sublevel graphs, which appears for a specific sublevel graph G_{α_b}, the value α_b, called the *birth time* of this connected component, being the function value at the tip of the branch. Since this connected component eventually gets merged with another at the value α_d where the branch connects to the graph, the corresponding value $\alpha_d \geq \alpha_b$ is stored and called the *death time* of the branch, and one says that the branch persists on the interval $[\alpha_b, \alpha_d]$. Similarly, each time a loop of G appears in a specific sublevel graph, the corresponding α value is saved. Note however that loops persist until $\alpha = +\infty$ since loops never disappear from the sequence of sublevel graphs, and the same applies for whole connected components of G. Moreover, branches pointing upwards are completely missed, since they do not create connected components when they appear in the sublevel graphs, making ordinary persistence unable to detect them.

Extended persistence. To handle this issue, extended persistence refines the analysis by also looking at the *superlevel graphs* $G^\alpha = (V^\alpha, E^\alpha)$ where $\alpha \in \mathbb{R}$, $V^\alpha = \{v \in V \,:\, f(v) \geq \alpha\}$ and $E^\alpha = \{(v_1, v_2) \in E \,:\, v_1, v_2 \in V^\alpha\}$. Similarly, making α decrease from $+\infty$ to $-\infty$ gives a sequence of decreasing subgraphs, for which structural changes can also be recorded, as illustrated on the bottom row of Figure 1. In particular, death times can be defined for loops and whole connected components by picking the superlevel graphs for which the feature appears again, and using the corresponding α value as the death time for these features. Moreover, branches pointing upwards can be detected in this sequence of superlevel graphs, in the exact same way that downwards branches were in the sublevel graphs. See the upper part of Figure 2. Finally, the family of intervals of the form $[\alpha_b, \alpha_d]$ is turned into a multiset of points (i.e. point cloud where points are counted with multiplicity) in the Euclidean plane $\mathbb{R}^2$ by using the interval bounds as coordinates. This multiset is called the *extended persistence diagram* of f and is denoted by $\mathrm{Dg}(G, f)$. Since graphs have four topological features, namely the upwards and downwards branches, the loops and the connected components, points in extended persistence diagrams can have four different types, that are denoted as Ord_0, Rel_1, Ext_0^+ and Ext_1^- for downwards branches, upwards branches, connected components and loops respectively:

$$\mathrm{Dg}(G, f) = \mathrm{Ord}_0(G, f) \sqcup \mathrm{Rel}_1(G, f) \sqcup \mathrm{Ext}_0^+(G, f) \sqcup \mathrm{Ext}_1^-(G, f).$$

See the lower part of Figure 2. Note that an advantage of using extended persistence is that all diagram types can be treated similarly, in the sense that points in each type all have finite coordinates, contrarily to ordinary persistence for which points have to be divided into two groups, with either finite or infinite death times, and thus processed differently, as in [15]. In practice, computing extended persistence diagrams can be efficiently achieved with the C++/Python `Gudhi` library [23].

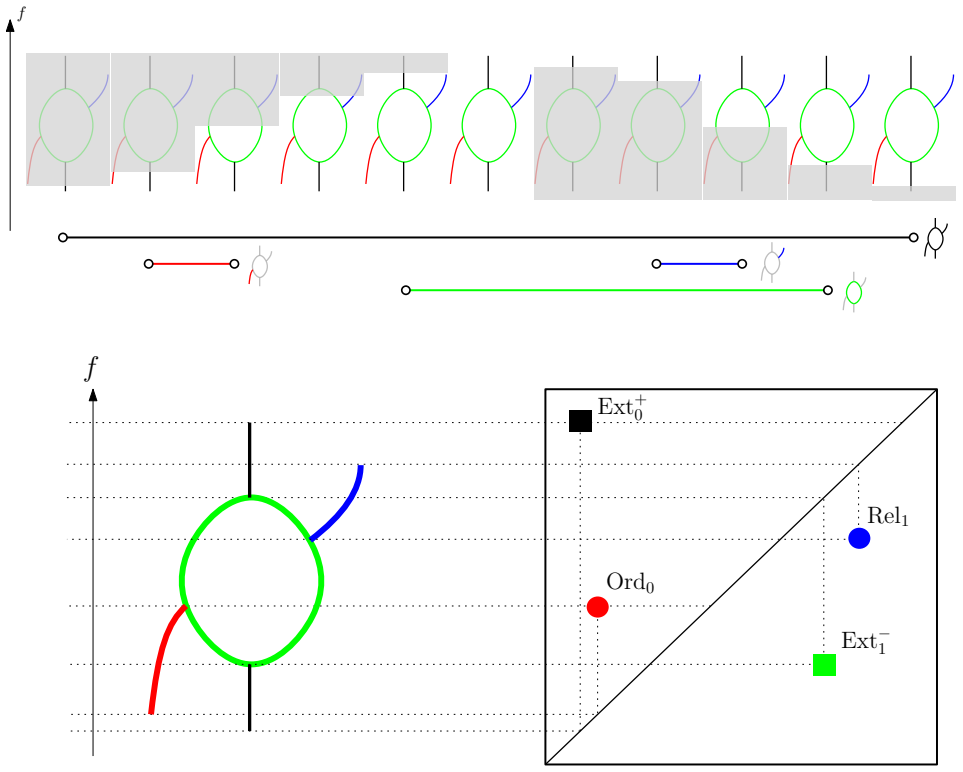

Figure 2: Extended persistence diagram computed on a graph. Each topological feature of the graph is detected in the sequence of sublevel and superlevel graphs shown at the top of the figure. The corresponding intervals are displayed under the sequence, and the extended persistence diagram given by the intervals is shown at the bottom of the figure.

Choosing the function f used to generate extended persistence diagrams is thus a critical step of our pipeline, which we detail, in the context of graph classification, in Section 3. Note however that the architecture that we propose subsequently in Section 4 does not make any assumptions on the choice of function f, and can be used to process any set of (extended) persistence diagrams. Before moving to the next section, we finally recall an important property that (extended) persistence diagrams enjoy, namely their *stability*.

Stability. The space of persistence diagrams is generally equipped with a metric called the *bottleneck distance*, denoted by d_B. Its proper definition is not required for this work and can be found in [13, Ch. VIII.2]. An important property of extended persistence diagrams endowed with the bottleneck distance is their *stability* with respect to perturbations applied to the functions generating them. Indeed, diagrams computed with similar functions are also going to be similar in a Lipschitz-continuous manner, as stated in the following theorem:

Theorem 2.1 ([9, 12, 20]) *Let $G = (V, E)$ be a graph and $f, g : V \to \mathbb{R}$ be two functions defined on its vertices. Then:*

$$\mathrm{d}_B(\mathrm{Dg}(G, f), \mathrm{Dg}(G, g)) \leq \|f - g\|_\infty, \tag{1}$$

where d_B stands for the so-called bottleneck distance *between persistence diagrams. Moreover, this inequality is also satisfied for each of the subtypes $\mathrm{Ord}_0, \mathrm{Rel}_1, \mathrm{Ext}_0^+$ and Ext_1^- individually.*

Despite its appealing theoretical properties, the bottleneck distance is not suited for statistical and learning applications of persistence diagrams. The space of persistence diagrams equipped with the bottleneck distance does not have a linear structure, and computing the bottleneck distance for large diagrams can be computationally prohibitive. As such, performing machine learning with persistence diagrams requires more investigation. For instance, in practical applications, diagrams are generally embedded into Hilbert spaces of either finite [2, 8] or infinite dimension [6, 18, 19, 21]

using kernel methods. In Section 4, we follow the former approach by building a versatile layer for neural networks that learn task-specific vectorizations for (extended) persistence diagrams.

Remark 2.2 *Note that, even though extended persistence diagrams are stable, they are in fact bag-of-features descriptors, since they encode the presence and size of every topological feature but are oblivious to their actual layout in the graphs. Hence, different graphs might still have same extended persistence diagrams. However, it has been shown that extended persistence diagrams are actually complete descriptors for graphs that are close enough [7].*

3 Heat kernel signatures on graphs

We have seen in Section 2 how extended persistence diagrams can be computed from a graph and a function defined on its vertices. Thus, in this section, we provide more details about the family of functions that we use in our experiments in Section 5, called the *heat kernel signatures* (HKS). HKS is an example of spectral family of signatures, i.e. functions derived from the spectral decomposition of graph Laplacians, which provide informative features for graph analysis. Although our approach based on extended persistence diagrams can be used with any family of signatures, we restrict for clarity of exposure to the HKS, whose extended persistent homology turns out to be stable with respect to the edges weights, leading to robust signatures that we eventually feed to our neural network in Sections 4 and 5.

Graph Laplacian. The adjacency matrix A of a graph G with vertex set $V = \{v_1, \cdots, v_n\}$ is the symmetric $n \times n$ matrix defined by $A_{i,j} = w_G(v_i, v_j)$ if $(v_i, v_j) \in E$ and $A_{i,j} = 0$ otherwise. The degree matrix D is the diagonal matrix defined by $D_{i,i} = \sum_j w_G(v_i, v_j)$. The normalized graph Laplacian $L_w = L_w(G)$ is the linear operator acting on the space of functions defined on the vertices of G which is represented by the matrix $L_w = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$. It admits an orthonormal basis of eigenfunctions $\Phi = \{\phi_1, \cdots, \phi_n\}$ and its eigenvalues satisfy $0 \leq \lambda_1 \leq \cdots \leq \lambda_n \leq 2$.

Heat kernel signatures. Note that, as the orthonormal eigenbasis Φ is not uniquely defined, the eigenfunctions ϕ_i cannot be directly used as signatures to compare graphs. To overcome this problem we consider the *heat kernel signatures*:

Definition 3.1 ([16, 22]) *Let $h(\lambda, t) = \exp(-t\lambda)$. Then, given a graph $G = (V, E)$ and $t \geq 0$, the* heat kernel signature *at t is the function $\mathrm{hks}_{G,t}$ defined on V by:*

$$\mathrm{hks}_{G,t} : v \mapsto \sum_{k=1}^{n} h(\lambda_k, t)\phi_k(v)^2.$$

Remark 3.2 *Note that the HKS are part of a more general family of signatures, the* Laplacian family signatures *[16], in which h can be replaced by any other function. Included in this general family are, for instance, the wave kernel signatures [3] and the wavelet kernel signatures [14]. The constructions and results of the present work extend straightforwardly to these other signatures.*

Remark 3.3 *Since the HKS can also be computed by taking the diagonal of the heat kernel associated to the heat equation on the graph G—see, e.g., [11, Chapter 10]—they can be seen as multiscale signatures and they inherit properties that make them particularly well-suited for graph analysis and classification. In particular, they do not depend on the choice of orthonormal eigenbasis Φ and are invariant under graph isomorphisms: if $\pi : G \to G'$ is a graph isomorphism, then for any $v \in V$, one has $\mathrm{hks}_{G',t}(\pi(v)) = \mathrm{hks}_{G,t}(v)$.*

Stability. The HKS have already been used as signatures to address graph matching problems [16] or to define spectral descriptors to compare graphs [25]. These signatures rely on the distributions of values taken by the HKS but not on their global topological structures, which are encoded in their extended persistence diagram. Combining Theorem 2.1 for extended persistence diagrams with stability properties of the HKS proven in [16] leads to the stability of extended persistence diagrams of the HKS under perturbations of the Laplacian, as stated in the following theorem.

Theorem 3.4 *Let $t \geq 0$ and let L_w be the Laplacian matrix of a graph G with n vertices. Let $\tilde{G}$ be another graph with n vertices and Laplacian matrix $\tilde{L}_w = L_w + E$. Then there exists a constant $C(G,t) > 0$ only depending on t and the spectrum of L_w such that:*

$$\mathrm{d}_B(\mathrm{Dg}(G, \mathrm{hks}_{G,t}), \mathrm{Dg}(G, \mathrm{hks}_{\tilde{G},t})) \leq C(G,t)\|E\|,$$

as soon as $\|E\|$, the Frobenius norm of E, is small enough.

4 Neural Network for Persistence Diagrams

In this section, we detail the general neural network architecture that we use to perform classification on the extended persistence diagrams generated with HKS that we presented in Sections 2 and 3. To do so, we leverage a recent neural network architecture called DeepSet [28] that was developed to handle sets of points.

4.1 DeepSet neural network

The main purpose of the DeepSet architecture is to be *invariant* to the point orderings in the sets. Any such neural network is called a *permutation invariant network*. In order to achieve this, Zaheer et al. [28] proposed to develop a network implementing the general equation:

$$F(X) = \rho\left(\sum_{i=1}^{n} \phi(x_i)\right), \tag{2}$$

where $F : \mathbb{R}^{n\times p} \to \mathbb{R}^d$ is the function implemented by the network (usually $d = 1$ if one is solving a regression problem and d is equal to the number of classes if one is solving a classification problem), $X = \{x_1, \cdots, x_n\} \subset \mathbb{R}^p$ is a set with n points, $\phi : \mathbb{R}^p \to \mathbb{R}^q$ is a point transformation and $\rho : \mathbb{R}^q \to \mathbb{R}^d$ is a final transformation often used to make the input and output dimensions agree. Actually, it is shown in [28, Theorem 2] that if the cardinality n is the same for all sets, then for any permutation invariant network F, there exist ρ_F and ϕ_F such that Equation (2) is satisfied. Moreover, this is still true for variable n if the sets belong to some countable space.

4.2 Extension to persistence diagrams

DeepSet for persistence diagrams. In this section, we slightly extend the DeepSet architecture to persistence diagrams and we show how several approaches for vectorizing persistence diagrams can be seen as special cases of our general neural network. Our general architecture implements the following general form:

$$F(\mathrm{Dg}) = \rho\left(\mathrm{op}\left(\{w(p)\cdot\phi(p)\}_{p\in\mathrm{Dg}}\right)\right), \tag{3}$$

where op is any permutation invariant operation (such as minimum, maximum, sum, kth largest value...), $w : \mathbb{R}^2 \to \mathbb{R}$ is a weight function for the persistence diagram points, and $\phi : \mathbb{R}^2 \to \mathbb{R}^q$ is a point transformation. Let us now discuss potential choices for these functions that we implemented for our experiments in Section 5.

Weight function w. In our experiments, w is actually treated as a parameter whose values are optimized during training. More precisely, the diagrams are scaled to $[0,1]\times[0,1]$, so that w becomes a function defined on the unit square, that we approximate by discretizing this square into a set of pixels. The value of w on each pixel is then optimized individually during training.

Point transformation ϕ and vectorization of persistence diagrams. We now present three point transformations that we used and implemented for parameter ϕ in Equation (3). Then, we explicit the persistence diagram vectorizations of the literature that are induced by each of these point transformations.

- **Triangle function.** The triangle function Λ_p associated to a point $p = (x, y) \in \mathbb{R}^2$ is:

$$\Lambda_p : t \mapsto \max\{0, y - |t - x|\}.$$

Let $q \in \mathbb{N}$ and $t_1, \cdots, t_q \in \mathbb{R}$. The *triangle point transformation* $\phi_\Lambda : \mathbb{R}^2 \to \mathbb{R}^q$ is:

$$\phi_\Lambda : p \mapsto \begin{bmatrix} \Lambda_p(t_1) \\ \Lambda_p(t_2) \\ \vdots \\ \Lambda_p(t_q) \end{bmatrix}$$

See Figure 3 for an example.

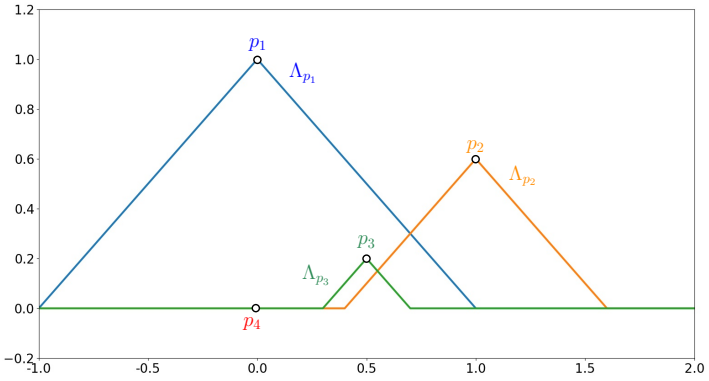

Figure 3: Example of triangle functions on a persistence diagram with four points p_1, p_2, p_3, p_4. Note that Λ_{p_4} is not displayed since it is the null function.

- **Gaussian function.** The Gaussian function Γ_p associated to a point $p = (x, y) \in \mathbb{R}^2$ is:

$$\Gamma_p : t \mapsto \exp\left(-\frac{\|p - t\|_2^2}{2\sigma^2}\right).$$

Let $q \in \mathbb{N}$ and $t_1, \cdots, t_q \in \mathbb{R}^2$. The *Gaussian point transformation* $\phi_\Gamma : \mathbb{R}^2 \to \mathbb{R}^q$ is:

$$\phi_\Gamma : p \mapsto \begin{bmatrix} \Gamma_p(t_1) \\ \Gamma_p(t_2) \\ \vdots \\ \Gamma_p(t_q) \end{bmatrix}$$

See Figure 4 for an example.

- **Line function.** The line function L_Δ associated to a line Δ with direction vector $e_\Delta \in \mathbb{R}^2$ and bias $b_\Delta \in \mathbb{R}$ is:

$$L_\Delta : p \mapsto \langle p, e_\Delta \rangle + b_\Delta.$$

Let $q \in \mathbb{N}$ and $\Delta_1, \cdots, \Delta_q$ be q lines. The *line point transformation* $\phi_L : \mathbb{R}^2 \to \mathbb{R}^q$ is:

$$\phi_L : p \mapsto \begin{bmatrix} L_{\Delta_1}(p) \\ L_{\Delta_2}(p) \\ \vdots \\ L_{\Delta_q}(p) \end{bmatrix}$$

See Figure 5 for an example.

Note that line point transformations are actually examples of so-called *permutation equivariant functions*, which are specific functions defined on sets of points that were used to build the DeepSet architecture in [28].

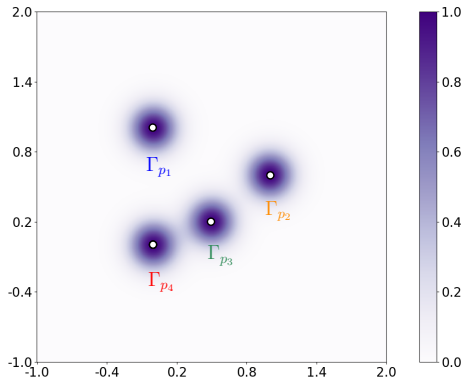

Figure 4: Example of Gaussian functions on a persistence diagram with four points p_1, p_2, p_3, p_4.

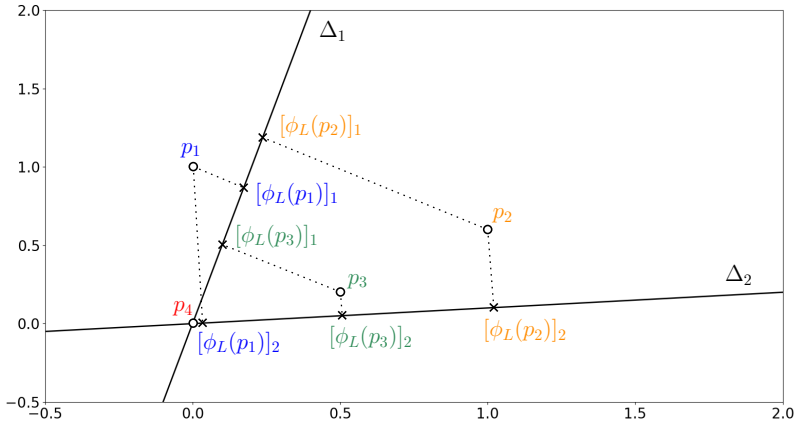

Figure 5: Example of line functions for two lines Δ_1, Δ_2 on a persistence diagram with four points p_1, p_2, p_3, p_4. We use $[\cdot]_k$ to denote the kth coordinate of a vector.

Relation to existing vectorization methods. Here we explicit the connections between our generic formula in Equation (3) and various vectorization and kernel methods for persistence diagrams in the literature.

- Using $\phi = \phi_\Lambda$ with samples $t_1, \cdots, t_q \in \mathbb{R}$, op $=$ kth largest value, $w(p) = 1$, amounts to evaluating the kth *persistence landscape* [4] on $t_1, \cdots, t_q \in \mathbb{R}$.
- Using $\phi = \phi_\Lambda$ with samples $t_1, \cdots, t_q \in \mathbb{R}$, op $=$ sum, arbitrary $w(p)$, amounts to evaluating the *persistence silhouette* weighted by w [10] on $t_1, \cdots, t_q \in \mathbb{R}$.
- Using $\phi = \phi_\Gamma$ with samples $t_1, \cdots, t_q \in \mathbb{R}^2$, op $=$ sum, arbitrary $w(p)$, amounts to evaluating the *persistence surface* weighted by w [2] on $t_1, \cdots, t_q \in \mathbb{R}^2$. Moreover, characterizing points of persistence diagrams with Gaussian functions is also the approach advocated in several kernel methods for persistence diagrams [18, 19, 21].
- Using $\phi = \phi_{\tilde{\Gamma}}$ where $\tilde{\Gamma}$ is a modification of the Gaussian point transformation defined with: $\tilde{\Gamma}_p = \Gamma_{\tilde{p}}$ for any $p = (x, y) \in \mathbb{R}^2$, where $\tilde{p} = p$ if $y \leq \nu$ for some $\nu > 0$, and $\left(x, \nu + \log\left(\frac{y}{\nu}\right)\right)$ otherwise, op $=$ sum, $w(p) = 1$, is the approach presented in [15]. This shows that we substantially generalize the architecture in this article.

- Using $\phi = \phi_L$ with lines $\Delta_1, \cdots, \Delta_q \in \mathbb{R}^2$, op $= k$th largest value, $w(p) = 1$, is similar to the approach advocated in [6], where the sorted projections of the points onto the lines are then compared with the $\|\cdot\|_1$ norm and exponentiated to build the so-called Sliced Wasserstein kernel for persistence diagrams.

Optimization. In practice, the samples $t_1, \cdots, t_q$ and lines $\Delta_1, \cdots, \Delta_q$ are parameters that are optimized on the training set by the network. This means that our approach looks for the best locations to evaluate the landscapes, silhouettes and persistence surfaces, as well as the best lines to project the diagrams onto, with respect to the problem the network is trying to solve.

5 Experimental results

In this section, we detail the experiments we used to validate our approach. Our code is based on the open source C++/Python library `Gudhi` [23], Python package `sklearn-tda` [5], and Python package `tensorflow` [1] and will be publicly released soon.

5.1 Datasets and data preprocessing

Datasets. We evaluate our classification model on a series of different graph datasets that are commonly used as a baseline in graph classification problems. A description of these datasets can be found in Table 2 in the Appendix.

- `REDDIT5K, REDDIT12K, COLLAB, IMDB-B, IMDB-M` are composed of social graphs. We obtain `REDDIT5K, REDDIT12K, COLLAB` from `http://www.mit.edu/~pinary/kdd/datasets.tar.gz`, and `IMDB-B, IMDB-M` are provided as `.graphml` files by [24].
- `BZR, COX2, DHFR, MUTAG, PROTEINS, NCI1, NCI109, FRANKENSTEIN` are composed of graphs coming from medical or biological frameworks. For all these datasets, we used the `.graphml` files provided by [24].

All the observations in these datasets are graphs represented by their adjacency matrices. Note that some of them (marked as * in Table 1) may have *attributes* on their nodes or edges, an additional information that our approach does not use.

Preprocessing. For each graph in each dataset, we generate a set of persistence diagrams along with some finite-dimensional features in the following way: for $t \in \{0.1, 1, 10, 100\}$, we compute the corresponding extended persistence diagram $\mathrm{Dg}(\mathrm{hks}_t)$ induced by HKS (see Section 2 and Section 3) with parameter t. We then divide each diagram into its four different types Ord_0, Rel_1, Ext_0^+ and Ext_1^- to get four different diagrams. Hence, each graph gives 16 persistence diagrams: one for each t parameter of the HKS and diagram type. These diagrams are then transformed with the birth-persistence function: $(x, y) \mapsto (x, y - x)$, truncated to the first k points which are the farthest away from the diagonal (we use a specific k for each dataset, see Table 3), and scaled to the unit square $[0, 1] \times [0, 1]$.

Additional features. We use in parallel feature vectors built on graphs, which contain the sorted eigenvalues of the normalized graph Laplacian L_w, along with the deciles of the HKS for each t.

5.2 Network architecture

In order to show the contribution of the extended persistence diagrams and the way we use them in neural networks with Equation (3), we voluntarily use a very simple network architecture. Namely, our network has only two layers: the first one consists of few channels that process the persistence diagrams in parallel. Note that we do not necessarily use all the diagrams produced in practice. Table 3 precises, for each experiments, which diagrams where used. The output of this layer is then concatenated with the graph features and the resulting vector is normalized. The second (and final) layer is a fully connected layer, whose output is used to make the prediction. See Figure 6.

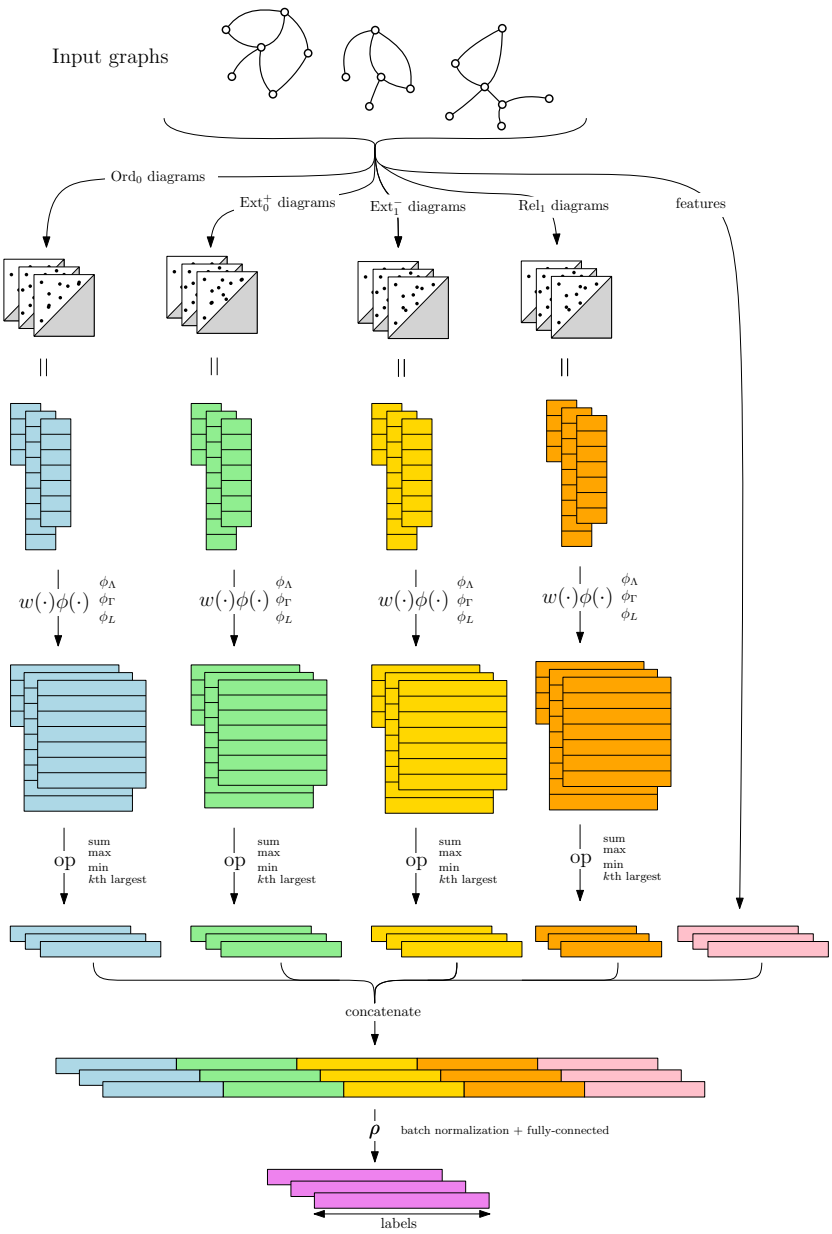

Figure 6: The network architecture used in our experiment for one given value of t.

Dataset	ScaleVariant [24]	DLtopo [15]	RetGK1 * [29]	RetGK11 * [29]	FGSD [26]	Spectral + HKS only	Spectral + HKS + TDA
REDDIT5K	—	54.5	56.1(±0.5)	55.3(±0.3)	47.8	49.7(±0.3)	**57.2**
REDDIT12K	—	44.5	**48.7**(±0.2)	47.1(±0.3)	—	39.7(±0.1)	48.2
COLLAB	—	—	81.0(±0.3)	80.6(±0.3)	80.0	67.8(±0.2)	74.6(±0.2)
IMDB-B	72.9	—	71.9(±1.0)	72.3(±0.6)	73.6	67.6(±0.6)	70.9(±0.7)
IMDB-M	50.3	—	47.7(±0.3)	48.7(±0.6)	52.4	44.5(±0.4)	47.2(±0.5)
BZR *	86.6	—	—	—	—	80.8(±0.8)	**87.2(±0.7)**
COX2 *	78.4	—	80.1(±0.9)	**81.4(±0.6)**	—	78.2(±1.3)	80.7(±1.2)
DHFR *	78.4	—	81.5(±0.9)	**82.5(±0.8)**	—	69.5(±1.0)	80.2(±0.9)
MUTAG *	88.3	—	90.3(±1.1)	90.1(±1.0)	**92.1**	85.8(±1.32)	91.2(±1.5)
PROTEINS *	72.6	—	**75.8(±0.6)**	75.2(±0.3)	73.4	73.5(±0.3)	75.3
NCI1 *	71.6	—	**84.5(±0.2)**	83.5(±0.2)	79.8	65.3(±0.2)	72.8(±0.3)
NCI109 *	70.5	—	—	—	78.8	64.9(±0.2)	70.6(±0.3)
FRANKENSTEIN	69.4	—	—	—	—	62.9(±0.1)	**70.7(±0.4)**

Table 1: Performance table. We report the mean accuracies and standard deviations given in each corresponding article. Note that for [15, 24, 26], mean accuracy is not averaged over ten 10-folds (single 10-fold for [15, 26] and average over 100 random splits at different proportions of training data for [24]) and thus not directly comparable to ours. * emphasizes that these datasets contain attributes (labels) on graph nodes and respectively, that a method leverages such attributes. Recall that our approach does *not* use any node label. Blue color represent the best method that does not use node labeling. Bold font represents the overall best score.

5.3 Experimental settings and results

Table 1 summaries the scores obtained by our approach on different sets of graphs. We compare it to four other graph classification methods:

- Scale-variant topo [24], which uses a kernel for persistence diagrams
- DLtopo [15], which uses a neural network for persistence diagrams. We recall from Section 4 that this approach is a specific case of ours.
- RetGK [29] is a graph classification method relying on a kernel approach that leverages *attributes* on the graph vertices and edges. It reaches state-of-the-art results on many datasets. Note that while the exact computation (denoted as RetGK1 in Table 1) can be quite long, this method can be efficiently approximated (RetGK11 in Table 1) while preserving good accuracy scores.
- FGDS [26] is a graph classification method that does not leverage attributes, and reaches state-of-the-art results on different datasets.

For each dataset, we compute a final score by averaging ten 10-folds (as in [29]), where a single 10-fold is computed by randomly shuffling the dataset, then splitting it into 10 different parts, and finally classifying each part using the nine others for training and averaging the classification accuracy obtained throughout the folds. We then report in Table 1 the average and standard deviation of these scores. One can see that, in most cases, our approach is comparable, if not better, than state-of-the-art results, despite avoiding kernels and using very simple neural networks architecture. More importantly, it can be observed from the last two columns of Table 1 that, for all datasets, the use of extended persistence diagrams significantly improves over using the additional features alone.

6 Conclusion

In this article, we propose a versatile, powerful and simple way to handle graphs in machine learning. More precisely, our approach is based on combining topological descriptors and neural networks: we presented a simple and easy way to encode graph structure into compact descriptors by combining an extension of persistence theory and a family of Laplacian-based graph functions. These descriptors, in addition to being provably robust, can then be fed to our simple and general neural network architecture that generalizes most of the techniques used to vectorize persistence diagrams that can be found in the literature. Finally, we validated our method by showing that it can achieve state-of-the-art results on several graph classification problems, despite being much simpler than most of its competitors.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Henry Adams, Tegan Emerson, Michael Kirby, Rachel Neville, Chris Peterson, Patrick Shipman, Sofya Chepushtanova, Eric Hanson, Francis Motta, and Lori Ziegelmeier. Persistence images: a stable vector representation of persistent homology. *Journal of Machine Learning Research*, 18(8), 2017.

[3] Mathieu Aubry, Ulrich Schlickewei, and Daniel Cremers. The wave kernel signature: A quantum mechanical approach to shape analysis. In *IEEE International Conference on Computer Vision*, pages 1626–1633. IEEE, 2011.

[4] Peter Bubenik. Statistical topological data analysis using persistence landscapes. *Journal of Machine Learning Research*, 16(77):77–102, 2015.

[5] Mathieu Carrière. sklearn-tda: a scikit-learn compatible python package for machine learning and tda. `https://github.com/MathieuCarriere/sklearn_tda`, 2018.

[6] Mathieu Carrière, Marco Cuturi, and Steve Oudot. Sliced Wasserstein kernel for persistence diagrams. In *International Conference on Machine Learning*, volume 70, pages 664–673, jul 2017.

[7] Mathieu Carrière and Steve Oudot. Local equivalence and intrinsic metrics between Reeb graphs. In *International Symposium on Computational Geometry*, volume 77, pages 25:1–25:15, 2017.

[8] Mathieu Carrière, Steve Oudot, and Maks Ovsjanikov. Stable topological signatures for points on 3d shapes. In *Computer Graphics Forum*, volume 34, pages 1–12. Wiley Online Library, 2015.

[9] Frédéric Chazal, Vin de Silva, Marc Glisse, and Steve Oudot. *The structure and stability of persistence modules*. Springer International Publishing, 2016.

[10] Frédéric Chazal, Brittany Terese Fasy, Fabrizio Lecci, Alessandro Rinaldo, and Larry Wasserman. Stochastic convergence of persistence landscapes and silhouettes. *Journal of Computational Geometry*, 6(2):140–161, 2015.

[11] Fan Chung. *Spectral graph theory*. American Mathematical Society, 1997.

[12] David Cohen-Steiner, Herbert Edelsbrunner, and John Harer. Extending persistence using Poincaré and Lefschetz duality. *Foundations of Computational Mathematics*, 9(1):79–103, feb 2009.

[13] Herbert Edelsbrunner and John Harer. *Computational topology: an introduction*. American Mathematical Society, 2010.

[14] David Hammond, Pierre Vandergheynst, and Rémi Gribonval. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150, 2011.

[15] Christoph Hofer, Roland Kwitt, Marc Niethammer, and Andreas Uhl. Deep learning with topological signatures. In *Advances in Neural Information Processing Systems*, pages 1634–1644, 2017.

[16] Nan Hu, Raif Rustamov, and Leonidas Guibas. Stable and informative spectral signatures for graph matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2305–2312, 2014.

[17] Diederik Kingma and Jimmy Ba. Adam: a method for stochastic optimization. *arXiv*, dec 2014.

[18] Genki Kusano, Yasuaki Hiraoka, and Kenji Fukumizu. Persistence weighted Gaussian kernel for topological data analysis. In *International Conference on Machine Learning*, volume 48, pages 2004–2013, jun 2016.

[19] Tam Le and Makoto Yamada. Persistence Fisher kernel: a Riemannian manifold kernel for persistence diagrams. In *Advances in Neural Information Processing Systems*, pages 10027–10038, 2018.

[20] Steve Oudot. *Persistence theory: from quiver representations to data analysis*. American Mathematical Society, 2015.

[21] Jan Reininghaus, Stefan Huber, Ulrich Bauer, and Roland Kwitt. A stable multi-scale kernel for topological machine learning. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[22] Jian Sun, Maks Ovsjanikov, and Leonidas Guibas. A concise and provably informative multi-scale signature based on heat diffusion. *Computer graphics forum*, 28:1383–1392, 2009.

[23] The GUDHI Project. *GUDHI User and Reference Manual*. GUDHI Editorial Board, 2015.

[24] Quoc Hoan Tran, Van Tuan Vo, and Yoshihiko Hasegawa. Scale-variant topological information for characterizing complex networks. *arXiv preprint arXiv:1811.03573*, 2018.

[25] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, Alexander Bronstein, and Emmanuel Müller. Netlsd: hearing the shape of a graph. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2347–2356. ACM, 2018.

[26] Saurabh Verma and Zhi-Li Zhang. Hunt for the unique, stable, sparse and fast feature learning on graphs. In *Advances in Neural Information Processing Systems*, pages 88–98, 2017.

[27] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv*, oct 2018.

[28] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan Salakhutdinov, and Alexander Smola. Deep sets. In *Advances in Neural Information Processing Systems*, pages 3391–3401, 2017.

[29] Zhen Zhang, Mianzhi Wang, Yijian Xiang, Yan Huang, and Arye Nehorai. Retgk: Graph kernels based on return probabilities of random walks. In *Advances in Neural Information Processing Systems*, pages 3968–3978, 2018.

Dataset	Nb graphs	Nb classes	Av. nodes	Av. Edges	Av. β_0	Av. β_1
REDDIT5K	5000	5	508.5	594.9	3.71	90.1
REDDIT12K	12000	11	391.4	456.9	2.8	68.29
COLLAB	5000	3	74.5	2457.5	1.0	2383.7
IMDB-B	1000	2	19.77	96.53	1.0	77.76
IMDB-MULTY	1500	3	13.00	65.94	1.0	53.93
BZR	405	2	35.75	38.36	1.0	3.61
COX2	467	2	41.22	43.45	1.0	3.22
DHFR	756	2	42.43	44.54	1.0	3.12
MUTAG	188	2	17.93	19.79	1.0	2.86
PROTEINS	1113	2	39.06	72.82	1.08	34.84
NCI1	4110	2	29.87	32.30	1.19	3.62
NCI109	4127	2	29.68	32.13	1.20	3.64
FRANKENSTEIN	4337	2	16.90	17.88	1.09	2.07

Table 2: Datasets description. β_0 (resp. β_1) stands for the 0th-Betti-number (resp. 1st), that is the number of connected components (resp. cycles) in a graph. In particular, an average $\beta_0 = 1.0$ means that all graph in the dataset are connected, and in this case $\beta_1 = \#\{\text{edges}\} - \#\{\text{nodes}\}$.

A Datasets description

Table 2 summarizes key information for each dataset.

B Parameters used in our experiments

Input data was fed to the network with mini-batches of size 128. We give, for each dataset, parameters details (extended persistence diagrams, neural network architecture, optimizers, etc.) used to obtain the scores we present in Table 1 in the column **Our approach**. In Table 3, we use the following shortcuts:

- hks_t: extended persistence diagram obtained with HKS on the graph with parameter t.
- prom(k): preprocessing step consisting in keeping the k points that are the farthest away from the diagonal.
- channels are denoted in the following way:
  - Im(p, (a, b), q, op) stands for a function ϕ obtained by using a Gaussian point transformation ϕ_Γ sampled on $(p \times p)$ grid on the unit square followed by a convolution with a filters of size $b \times b$, for a weight function w optimized on a $(q \times q)$ grid and for an operation op.
  - Pm(d_1, d_2, q, op) stands for a function ϕ obtained by using a line point transformation ϕ_L with d_1 lines followed by a permutation equivariant function with parameters $A, B \in \mathbb{R}^{d_1 \times d_2}$, $C \in \mathbb{R}^{d_2}$ and $B_X = \max(X \cdot B)$, where the maximum is over each column, for a weight function w optimized on a $(q \times q)$ grid and for an operation op.
- adam(λ, d, e) stands for the ADAM optimizer [17] with learning rate λ, using an Exponential Moving Average[1] with decay rate d, and run during e epochs.

[1] `https://www.tensorflow.org/api_docs/python/tf/train/ExponentialMovingAverage`

Dataset	Func. used	PD preproc.	DeepSet Channel	Optim.
REDDIT5K	$\mathrm{hks}_{1.0}$	prom(300)	Pm(25,25,10,sum)	adam(0.01, 0.99, 70)
REDDIT12K	$\mathrm{hks}_{1.0}$	prom(400)	Pm(5,5,10,sum)	adam(0.01, 0.99, 400)
COLLAB	$\mathrm{hks}_{0.1}$, hks_{10}	prom(200)	Im(20,(10,2),20,sum)	adam(0.01, 0.9, 500)
IMDB-B	$\mathrm{hks}_{0.1}$, hks_{10}	prom(200)	Im(20,(10,2),20,sum)	adam(0.01, 0.9, 500)
IMDB-M	$\mathrm{hks}_{0.1}$, hks_{10}	prom(200)	Im(20,(10,2),20,sum)	adam(0.01, 0.9, 500)
BZR	$\mathrm{hks}_{0.1}$, hks_{10}	—	Im(15,(10,2),10,sum)	adam(0.01, 0.9, 300)
COX2	hks_{10}	—	Im(20,(10,2),20,sum)	adam(0.01, 0.9, 300)
DHFR	hks_{10}	—	Im(20,(10,2),20,sum)	adam(0.01, 0.9, 300)
MUTAG	hks_{10}	—	Im(20,(10,2),20,sum)	adam(0.01, 0.9, 300)
PROTEINS	hks_{10}	prom(200)	Im(15,(10,2),10,sum)	adam(0.01, 0.9, 300)
NCI1	hks_{10}	—	Im(20,(10,2),20,sum)	adam(0.01, 0.9, 300)
NCI109	hks_{10}	—	Im(20,(10,2),20,sum)	adam(0.01, 0.9, 300)
FRANKENSTEIN	hks_{10}	—	Im(20,(10,2),20,sum)	adam(0.01, 0.9, 300)

Table 3: Settings used to generate our experimental results.