PersLay: A Simple and Versatile Neural Network Layer for Persistence Diagrams

Mathieu Carrière
Rabadan 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

Persistence diagrams, a key descriptor from Topological Data Analysis, encode and summarize all sorts of topological features and have already proved pivotal in many different applications of data science. But persistence diagrams are weakly structured and therefore constitute a difficult input for most Machine Learning techniques. To address this concern several vectorization methods have been put forward that embed persistence diagrams into either finite-dimensional Euclidean spaces or implicit Hilbert spaces with kernels. But finite-dimensional embeddings are prone to miss a lot of information about persistence diagrams, while kernel methods require the full computation of the kernel matrix.

We introduce PersLay: a simple, highly modular layer of learning architecture for persistence diagrams that allows to exploit the full capacities of neural networks on topological information from any dataset. This layer encompasses most of the vectorization methods of the literature. We illustrate its strengths on challenging classification problems on dynamical systems orbit or real-life graph data, with results improving or comparable to the state-of-the-art. In order to exploit topological information from graph data, we show how graph structures can be encoded in the so-called extended persistence diagrams computed with the heat kernel signatures of the graphs.

1 Introduction

Topological Data Analysis is a field of data science whose purpose is to capture and encode the topological features (such as the connected components, loops, cavities...) that are present in datasets in order to improve inference and prediction. Its main descriptor is the so-called *persistence diagram*, which takes the form of a set of points in the Euclidean plane $\mathbb{R}^2$, each point corresponding to a topological feature of the data. This descriptor has been successfully used in many different applications of data science, such as material science [4], signal analysis [24], cellular data [5], or shape recognition [22] to name a few. This wide range of applications is mainly due to the fact that persistence diagrams encode information whose essence is topological, and as such this information tends to be complementary to the one captured by more classical descriptors.

However, the space of persistence diagrams heavily lacks structure: different persistence diagrams may have different number of points, and several basic operations are not well-defined, such as

Preprint. Under review.

addition and mean. This dramatically impedes their capacity to be used in machine learning applications so far. To handle this issue, a lot of attention has been devoted to the *vectorization* of persistence diagrams through the construction of either *finite-dimensional embeddings* of persistence diagrams [2, 8, 11, 18], i.e., embeddings turning diagrams into vectors in Euclidean space $\mathbb{R}^d$, or *kernels* for persistence diagrams [3, 9, 20, 21, 25], i.e., generalized scalar products that implicitly turn diagrams into elements of infinite-dimensional Hilbert spaces. Unfortunately, both approaches suffer from major issues: it has been shown that finite-dimensional embeddings miss a lot of information about persistence diagrams [7], and kernel methods require to compute and store the kernel evaluations for each pair of persistence diagrams. Since all available kernels have a complexity that is at least linear, and often quadratic, in the number of persistence diagram points for a single matrix entry computation, kernel methods quickly become very expensive in terms of running time and memory usage on large sets or for large diagrams. An interesting preliminary approach has been developed in trying to feed persistence diagrams to neural networks [16], which consists in adapting and branching an existing vectorization method (the so-called *persistence images* [2]) to a large neural network. However, there are applications for which persistence images might be inappropriate a choice, and it is generally hard to tell ahead of time which kernel or vectorization method will be the most relevant one.

In this article, we present a framework to make full use of the modularity, learning capacities and computing power of *neural networks* for Topological Data Analysis. Building on the recent introduction of *DeepSet* from [33] that produces a neural network formatting targeted at processing sets of points, we apply and extend that framework to the context of persistence diagrams, thus defining *PersLay*: a simple, highly versatile, automatically differentiable layer for neural network architectures that can process topological information from all sorts of datasets. Our framework encompasses most of the popular vectorization and kernel methods that exist in the literature. We demonstrate its strengths on two challenging applications where we reach, improve or significantly improve over state-of-the-art classifying accuracies with performances evaluated by simple network architectures based on PersLay layers. The first is a large-scale classification application on a synthetic set of a hundred thousand dynamical system orbits. Second, we turn to a real graph classification application with benchmark datasets borrowing from the fields of biology, chemistry or social networks. Since graph data also suffer from a lack of general structure, we independently provide a robust method to efficiently encode graph information in a theoretically sound way using an extension of ordinary persistence called *extended persistence*, that is computed from a family of functions defined on the graph vertices called *heat kernel signatures* with ensuing stability properties. Lastly, we make our contribution available as a public `tensorflow`-based Python package.

To summarize our contributions:

- We introduce a simple, versatile neural network layer for persistence diagrams, called PersLay for handling topological information from all sorts of datasets, encompassing most of vectorizing approaches in the literature.
- We showcase the strength of our method by achieving state-of-the-art effectiveness on classifying a collection of a hundred thousand synthetic orbit data from dynamical systems, as well as on difficult graph classification problems from the literature.
- We introduce a theoretically-based approach for extracting topological information from graph data using a combination of *extended persistence* and *graph signatures*.
- We provide a ready-to-use PersLay Python package based on `tensorflow`: `https://github.com/MathieuCarriere/perslay`.

The basics of (extended) persistence theory are first introduced in Section 2. Our general neural network layer PersLay for persistence diagrams is then defined in Section 3. Finally, the two applications showcased are presented in Section 4.1 for the dynamical system problem, and in Section 4.2 for the graph application. This Section 4.2 also informs on the family of functions used to generate persistence diagrams, the so-called *heat kernel signatures* with stability properties.

2 Persistence diagrams as topological features

In this section, we recall the basics of persistence (and extended persistence) diagrams. We point to [13, 14, 23] for a thorough description of general (extended) persistence theory for metric spaces.

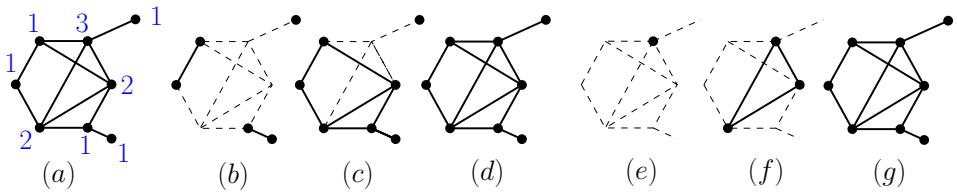


Figure 1: Illustration of sublevel and superlevel graphs. (a) Input graph (V, E) along with the values of a function $f : V \to \mathbb{R}$ (blue). (b, c, d) Sublevel graphs for $\alpha = 1, 2, 3$ respectively. (e, f, g) Superlevel graphs for $\alpha = 3, 2, 1$ respectively.

Ordinary persistence. Let us consider a pair (X, f), where X is a topological space, and $f\colon X \to \mathbb{R}$ is a real-valued function, and define the *sublevel set* $X_\alpha = \{x \in X \,:\, f(x) \leq \alpha\}$. Making α increase from $-\infty$ to $+\infty$ gives an increasing sequence of sets, called the *filtration* induced by f, and which starts with the empty set and ends with the whole space X. Ordinary persistence will record the time of appearance and disappearance of topological components (connected components, loops, cavities, etc.) in this sequence. For instance, one can record the value α_b for which a new connected component appears in X_{α_b}, called the *birth time* of the connected component. This connected component eventually gets merged with another for some value $\alpha_d \geq \alpha_b$, and thus α_d is stored and called the *death time* of the component, and one says that the component *persists* on the interval $[\alpha_b, \alpha_d]$. Similarly, we save the $[\alpha_b, \alpha_d]$ values of each loop, cavity, etc. that appears in a specific sublevel set X_{α_b} and disappears (get "filled") in X_{α_d}. This family of intervals is called the barcode, or *persistence diagram*, of (X, f), and can be represented as a multiset of points (i.e., point cloud where points are counted with multiplicity) supported on $\mathbb{R}^2$ with coordinates $\{(\alpha_b, \alpha_d)\}$.

In some context, ordinary persistence might not be sufficient to encode the topology of an object X. For instance, consider a graph $G = (V, E)$, with vertices V and (non-oriented) edges E. Let $f\colon V \to \mathbb{R}$ be a function defined on its vertices, and consider 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 $(b - d)$ in Figure 1. In this sequence $(G_\alpha)_\alpha$, loops persist forever since they never disappear from the sequence of sublevel graphs (they never get "filled"), and the same applies for whole connected components of G. Moreover, branches pointing upwards (with respect to the orientation given by f, see Figure 2) are missed (while those pointing downward are detected), 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 the issue stated above, extended persistence refines the analysis by also looking at the *superlevel set* $X^\alpha = \{x \in X \,:\, f(x) \geq \alpha\}$. Similarly, making α decrease from $+\infty$ to $-\infty$ also gives a sequence of increasing subsets, for which structural changes can be recorded.

Although extended persistence can be defined for general metric spaces (see the references given above), we restrict ourselves to the case where $X = G$ is a graph. The sequence of increasing superlevel graphs G^α is illustrated in Figure 1 $(e - g)$. 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. In this case, 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 Figure 2 for an illustration. Finally,

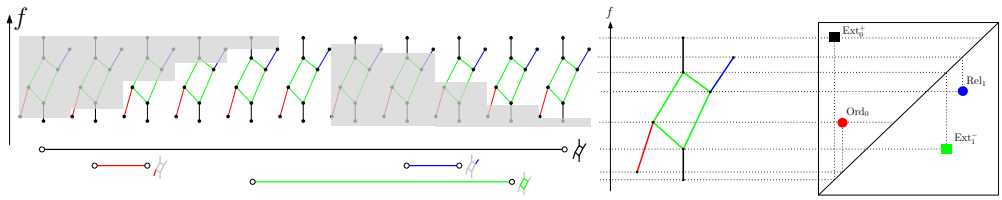


Figure 2: Extended persistence diagram computed on a graph: topological features of the graph are detected in the sequence of sublevel and superlevel graphs shown on the left of the figure. The corresponding intervals are displayed under the sequence: the black interval represents the connected component of the graph, the red one represents its downward branch, the blue one represents its upward branch, and the green one represents its loop. The extended persistence diagram given by the intervals is shown on the right.

the family of intervals of the form $[\alpha_b, \alpha_d]$ is turned into a multiset of points in the Euclidean plane $\mathbb{R}^2$ by using the interval endpoints as coordinates. This multiset is called the *extended persistence diagram* of f and is denoted by $\mathrm{Dg}(G, f) \subset \mathbb{R}^2$.

Since graphs have four types of topological features (see Figure 2), namely upwards branches, downwards branches, loops and connected components, the corresponding points in extended persistence diagrams can be of four different types. These types are denoted as Ord_0, Rel_1, Ext_0^+ and Ext_1^- for downwards branches, upwards branches, connected components and loops respectively:

$$\mathrm{Dg}(G, f) = \mathrm{Ord}_0(G, f) \sqcup \mathrm{Rel}_1(G, f) \sqcup \mathrm{Ext}_0^+(G, f) \sqcup \mathrm{Ext}_1^-(G, f). \tag{1}$$

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. Moreover, as each point represents a topological feature that persists along a given interval, $\mathrm{Dg}(G, f)$ provides explainable information about the topological structure of the pair (G, f). In practice, computing extended persistence diagrams can be efficiently done with the C++/Python `Gudhi` library [27]. Persistence diagrams are usually compared with the so-called bottleneck distance d_B—whose proper definition is not required for this work (see e.g. [14, VIII.2]). However, the resulting metric space is not Hilbert and as such, incorporating diagrams in a learning pipeline requires to design specific tools.

3 PersLay: A Neural Network Layer for Persistence Diagrams

We have seen in Section 2 how to derive topological descriptors from data, namely (extended) persistence diagrams, which we aim at using in machine learning applications. In this section, we present PersLay, our general neural network layer for (extended) persistence diagrams. To define it, we leverage a recent neural network architecture called DeepSet [33] that was targeted at processing sets of points.

The main purpose of the DeepSet design 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. [33] propose to process point sets with a layer implementing the general equation: $L(X) = \sum_{i=1}^{n} \phi(x_i)$, where $L : \mathbb{R}^{n \times p} \to \mathbb{R}^q$ is the function implemented by the layer, $X = \{x_1, \ldots, x_n\} \subset \mathbb{R}^p$, and $\phi : \mathbb{R}^p \to \mathbb{R}^q$ is a point transformation. Their final neural network architecture is then obtained by composing L with a transformation $\rho : \mathbb{R}^q \to \mathbb{R}^d$, which can be parametrized by any other neural network architecture. It is shown in [33, Theorem 2] that if the cardinality n is the same for all sets, then for any permutation invariant function F, there exist ρ_F and L_F such that $F = \rho_F \circ L_F$. Moreover, this is still true for variable n if the sets belong to some countable space.

In this work, we transpose this architecture to the context of persistence diagrams by defining and implementing a series of new permutation invariant layers that will generalize some standard tools used in Topological Data Analysis. To that end we define our generic neural network layer for persistence diagrams Dg, that we call PersLay, through the following equation:

$$\mathrm{PersLay}(\mathrm{Dg}) := \mathtt{op}\left(\{w(p) \cdot \phi(p)\}_{p \in \mathrm{Dg}}\right), \tag{2}$$

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 function that we call *point transformation function*. We emphasize that any neural network architecture ρ can be composed with PersLay to generate a neural network architecture for persistence diagrams. Let us now introduce the three point transformation functions that we use and implement for parameter ϕ in Equation (2).

- The *triangle point transformation* $\phi_\Lambda : \mathbb{R}^2 \to \mathbb{R}^q, p \mapsto [\Lambda_p(t_1), \Lambda_p(t_2), \ldots, \Lambda_p(t_q)]^T$ where the triangle function Λ_p associated to a point $p = (x, y) \in \mathbb{R}^2$ is $\Lambda_p \colon t \mapsto \max\{0, y - |t - x|\}$, with $q \in \mathbb{N}$ and $t_1, \ldots, t_q \in \mathbb{R}$.
- The *Gaussian point transformation* $\phi_\Gamma : \mathbb{R}^2 \to \mathbb{R}^q, p \mapsto [\Gamma_p(t_1), \Gamma_p(t_2), \ldots, \Gamma_p(t_q)]^T$, where the Gaussian function Γ_p associated to a point $p = (x, y) \in \mathbb{R}^2$ is $\Gamma_p \colon t \mapsto \exp\left(-\|p - t\|_2^2/(2\sigma^2)\right)$ for a given $\sigma > 0$, $q \in \mathbb{N}$ and $t_1, \ldots, t_q \in \mathbb{R}^2$.
- The *line point transformation* $\phi_L \colon \mathbb{R}^2 \to \mathbb{R}^q, p \mapsto \left[L_{\Delta_1}(p), L_{\Delta_2}(p), \ldots, L_{\Delta_q}(p)\right]^T$, where 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 $L_\Delta : p \mapsto \langle p, e_\Delta \rangle + b_\Delta$, with $q \in \mathbb{N}$ and $\Delta_1, \ldots, \Delta_q$ q lines.

Note that line point transformations are examples of *permutation equivariant functions*, which are specific functions defined on sets of points that were used to build the DeepSet layer in [33].

Formulation (2) has high representative power: it allows to remarkably encode most of classical persistence diagram representations with a very small set of point transformation functions ϕ, allowing to consider the choice of ϕ as a hyperparameter of sort. Let us show how we connect it to popular vectorizations and kernel methods for persistence diagrams in the literature.

- Using $\phi = \phi_\Lambda$ with samples $t_1, \dots, t_q \in \mathbb{R}$, $\texttt{op} = k$th largest value, $w(p) = 1$, amounts to evaluating the kth *persistence landscape* [3] on $t_1, \dots, t_q \in \mathbb{R}$.
- Using $\phi = \phi_\Lambda$ with samples $t_1, \dots, t_q \in \mathbb{R}$, $\texttt{op} =$ sum, arbitrary $w(p)$, amounts to evaluating the *persistence silhouette* weighted by w [11] on $t_1, \dots, t_q \in \mathbb{R}$.
- Using $\phi = \phi_\Gamma$ with samples $t_1, \dots, t_q \in \mathbb{R}^2$, $\texttt{op} =$ sum, arbitrary $w(p)$, amounts to evaluating the *persistence surface* weighted by w [2] on $t_1, \dots, t_q \in \mathbb{R}^2$. Moreover, characterizing points of persistence diagrams with Gaussian functions is also the approach advocated in several kernel methods for persistence diagrams [20, 21, 25].
- 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, $\texttt{op} =$ sum, $w(p) = 1$, is the approach presented in [16].
- Using $\phi = \phi_L$ with lines $\Delta_1, \dots, \Delta_q \in \mathbb{R}^2$, $\texttt{op} = k$th largest value, $w(p) = 1$, is similar to the approach advocated in [9], 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.

In practice, samples $t_1, \dots, t_q$ and lines $\Delta_1, \dots, \Delta_q$ 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.

4 Applications and experimental results

We now introduce two different applications aimed at showcasing both the scaling power, precision and versatility of PersLay. In order to truly showcase the contribution of this layer we use a very simple network architecture, namely a two-layer network. The first layer is the PersLay layer that processes persistence diagrams. The output of this layer is then either used as such (as in the first application §4.1) or concatenated with additional features (for the graph application §4.2). The resulting vector is normalized and fed to the second and final layer, a fully connected layer whose output is used for predictions. An illustration in the context of graph classification is found Fig. 3. We emphasize that this simplistic two-layer architecture is designed so as to produce knowledge and understanding, rather than the best possible performances.

In those applications, the weight function w from Equation (2) is actually treated as a trainable parameter whose values are optimized during training. More precisely, the input 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. As both ϕ and w are functions defined on $\mathbb{R}^2$, it suggests—we leave it as perspective work—that they can be interpreted and visualized in ways to help explain the network learning behavior on persistence diagrams.

Our results can be reproduced with the help of Table 5 in the supplementary material with specific settings for each experiment. The implementation relies on the open source C++/Python library `Gudhi` [27], Python packages `sklearn-tda` [6] and `tensorflow` [1], and is available at `https://github.com/MathieuCarriere/perslay`.

4.1 A large-scale dynamical system orbits application

The first application is demonstrated on synthetic data used as a benchmark in Topological Data Analysis [2, 9, 21]. It consists in sequences of points generated by different dynamical systems, see

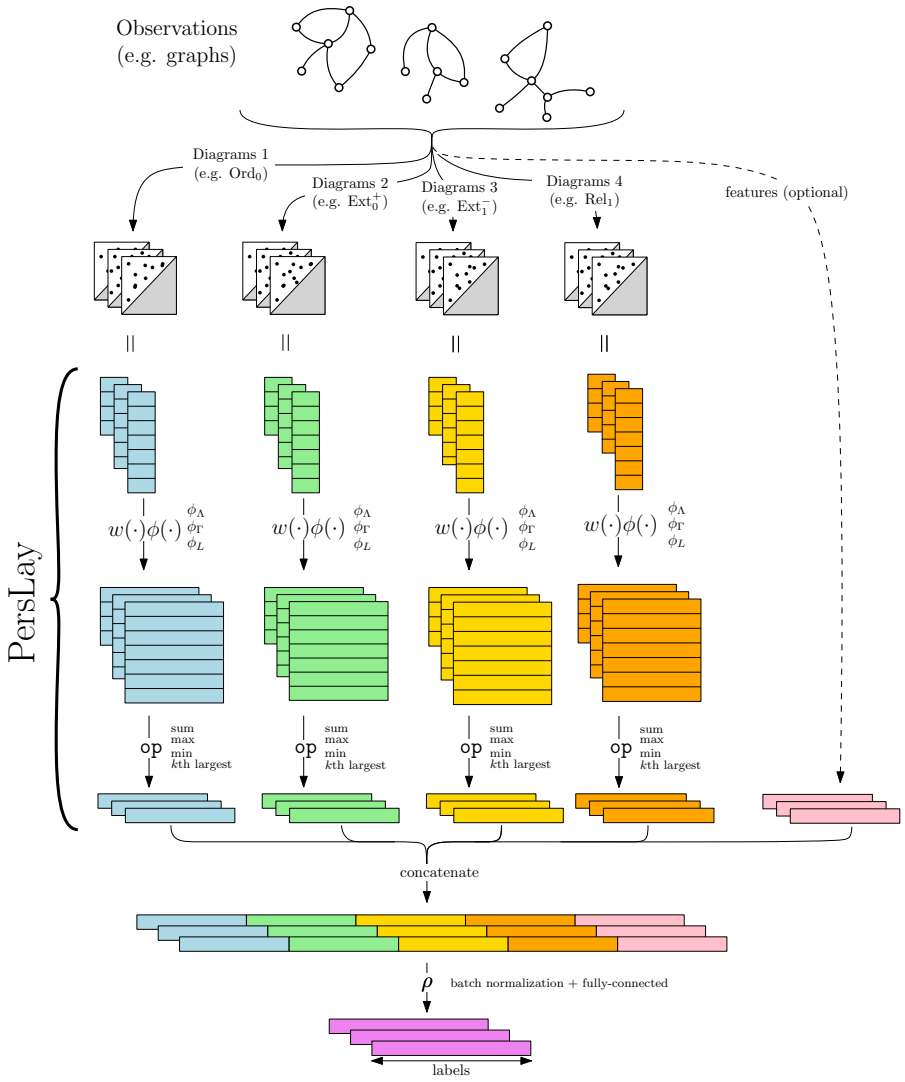


Figure 3: Network architecture illustrated in the case of our graph classification experiments (§4.2). Each graph is encoded as a set of persistence diagrams, then processed by an independent instance of PersLay. Each instance embeds diagrams in some vector space using two functions w, ϕ that are optimized during training and a fixed permutation-invariant operator op.

[15]. Given some initial position $(x_0, y_0) \in [0,1]^2$ and a parameter $r > 0$, we generate a point cloud $(x_n, y_n)_{n=1,\dots,N}$ following $x_{n+1} := x_n + r y_n(1-y_n)$ (mod 1) and $y_{n+1} := y_n + r x_{n+1}(1-x_{n+1})$ (mod 1). The orbits of this dynamical system are highly dependent on parameter r. More precisely, for some values of r, voids can form in these orbits (see Fig. 5 in the Supplementary Material), and as such, persistence diagrams are likely to perform well when attempting to classify orbits with respect to the value of r generating them. As in previous works [2, 9, 21], we use the five different parameters $r = 2.5, 3.5, 4.0, 4.1$ and 4.3 to simulate the different classes of orbits, with random initialization of (x_0, y_0) and $N = 1000$ points in each simulated orbit. These point clouds are then turned into persistence diagrams using a standard geometric filtration [10], namely the `AlphaComplex` filtration[1] in dimensions 0 and 1. Doing so, we generate two datasets. The first is `ORBIT5K`, where for each value of r, we generate $1,000$ orbits, ending up with a dataset of $5,000$ point clouds. This dataset is the same as the one used in [21]. The second, `ORBIT100K` contains $20,000$ orbits per class, resulting in a dataset of $100,000$ point clouds — a scale that kernel methods cannot handle. This dataset aims to show the edge of our neural-network based approach over kernels methods when dealing with very

[1] `http://gudhi.gforge.inria.fr/python/latest/alpha_complex_ref.html`

Dataset	PSS-K	PWG-K	SW-K	PF-K	PersLay
ORBIT5K	72.38($\pm$2.4)	76.63($\pm$0.7)	83.6($\pm$0.9)	85.9($\pm$0.8)	**87.7($\pm$1.0)**
ORBIT100K	—	—	—	—	**89.2($\pm$0.3)**

Table 1: Performance table. PSS-K, PWG-K, SW-K, PF-K stand for *Persistence Scale Space Kernel* [25], *Persistence Weighted Gaussian Kernel* [20], *Sliced Wasserstein Kernel* [9] and *Persistence Fisher Kernel* [21] respectively. We report the scores given in [21] for competitors on ORBIT5K, and the one we obtained using PersLay for both the ORBIT5K and ORBIT100K datasets.

large datasets of large diagrams. The goal is now to perform classification with the aforementioned architecture, while all the previous works dealing with this data [2, 9, 21] use a kernel method to classify the persistence diagrams built on top of the point clouds.

Results are displayed in Table 1. Not only do we improve on previous results for ORBIT5K, with performances on ORBIT100K we also show that classification accuracy is further increased as more observations are made available. For consistency we use the same accuracy metric as [21], we split observations in 70%-30% training-test sets and report the average test accuracy over 100 runs. The parameters used summarized Section C in the Supplementary Material.

4.2 PersLay for graph classification

We now specialize our framework to the evaluation of graph datasets. We have seen in Section 2 how extended persistence diagrams can be computed from a graph for a given function f, defined on its vertices. As the choice for this function is essential in capturing relevant topological information, we first provide more details about our choice of the family of such functions that we use in our experiments: the *heat kernel signatures* (HKS).

Heat kernel signatures on graphs. 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. The adjacency matrix A of a graph G with vertex set $V = \{v_1, \dots, v_n\}$ is the matrix $A := (\mathbf{1}_{(v_i,v_j)\in E})_{i,j}$. The degree matrix D is the diagonal matrix defined by $D_{i,i} = \sum_j A_{i,j}$. 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, and is represented by the matrix $L_w = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$. It admits an orthonormal basis of eigenfunctions $\Psi = \{\psi_1, \dots, \psi_n\}$ and its eigenvalues satisfy $0 \leq \lambda_1 \leq \cdots \leq \lambda_n \leq 2$. As the orthonormal eigenbasis Ψ is not uniquely defined, the eigenfunctions ψ_i cannot be used as such to compare graphs. Instead we consider the *heat kernel signatures*:

Definition 4.1 ([17, 26]) *Given a graph G and $t \geq 0$, the* heat kernel signature *at time t is the function* $\mathrm{hks}_{G,t}$ *defined on the vertices of G by* $\mathrm{hks}_{G,t} \colon v \mapsto \sum_{k=1}^{n} \exp(-t\lambda_k)\psi_k(v)^2$.

The HKS have already been used as signatures to address graph matching problems [17] or to define spectral descriptors to compare graphs [29]. 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 diagrams. Moreover the following theorem shows these diagrams to be stable with respect to the bottleneck distance d_B between persistence diagrams. The proof is found in the Supplementary Material, Section D.

Theorem 4.2 *Let $t \geq 0$ and let L_w be the Laplacian matrix of a graph G with n vertices. Let G' be another graph with n vertices and Laplacian matrix $\tilde{L}_w = L_w + W$. Then there exists a constant $C(G,t) > 0$ only depending on t and the spectrum of L_w such that, for small enough $\|W\|$:*

$$\mathrm{d}_B(\mathrm{Dg}(G, \mathrm{hks}_{G,t}), \mathrm{Dg}(G, \mathrm{hks}_{G',t})) \leq C(G,t)\|W\|, \tag{3}$$

Graph classification experiments. We are now ready to evaluate our architecture on a series of different graph datasets commonly used as a baseline in graph classification problems. REDDIT5K, REDDIT12K, COLLAB (from [32]) IMDB-B, IMDB-M (from [28]) are composed of social graphs. BZR, COX2, DHFR, MUTAG, PROTEINS, NCI1, NCI109, FRANKENSTEIN are graphs coming from medical or biological frameworks (also from [28]). A quantitative summary of these datasets is found in Table 4 from the Supplementary Material.

We compare performances with four other top graph classification methods. Scale-variant topo [28] uses a kernel for ordinary persistence diagrams computed on the graphs. RetGK [34] is a

Dataset	ScaleVariant[1]	RetGK1 * [2]	RetGK11 * [2]	FGSD [3]	GCNN [4]	Spectral + HKS [5]	PersLay
REDDIT5K	—	56.1(±0.5)	55.3(±0.3)	47.8	52.9	49.7(±0.3)	**56.6**(±0.3)
REDDIT12K	—	**48.7**(±0.2)	47.1(±0.3)	—	46.6	39.7(±0.1)	47.7(±0.2)
COLLAB	—	**81.0(±0.3)**	80.6(±0.3)	80.0	79.6	67.8(±0.2)	76.4(±0.4)
IMDB-B	72.9	71.9(±1.0)	72.3(±0.6)	**73.6**	73.1	67.6(±0.6)	70.9(±0.7)
IMDB-M	50.3	47.7(±0.3)	48.7(±0.6)	**52.4**	50.3	44.5(±0.4)	48.7(±0.6)
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)	**81.6(±1.0)**
DHFR *	78.4	81.5(±0.9)	**82.5(±0.8)**	—	—	69.5(±1.0)	81.8(±0.8)
MUTAG *	88.3	90.3(±1.1)	90.1(±1.0)	**92.1**	86.7	85.8(±1.3)	89.8(±0.9)
PROTEINS *	72.6	75.8(±0.6)	75.2(±0.3)	73.4	**76.3**	73.5(±0.3)	74.8(±0.3)
NCI1 *	71.6	**84.5(±0.2)**	83.5(±0.2)	79.8	78.4	65.3(±0.2)	72.8(±0.3)
NCI109 *	70.5	—	—	**78.8**	—	64.9(±0.2)	71.7(±0.3)
FRANKENSTEIN	69.4	—	—	—	—	62.9(±0.1)	**70.7(±0.4)**

Table 2: Mean accuracies and standard deviations over ten 10-folds, for [34][2] and our own evaluations (right hand side). In [28][1] the average accuracy over 100 random splits at different proportions of training data is reported, and in [30][3] and [31][4] the mean accuracy over a single 10-fold is reported. The * indicates those datasets that contain attributes (labels) on graph nodes and symmetrically the methods that leverage such attributes for classification purposes. Blue color represents the best method that does not use node attributes (when there are), and bold fonts mark the overall best score on a given problem.

kernel method for graphs that leverages *attributes* on the graph vertices and edges, and reaches state-of-the-art results on many datasets. Note that while the exact computation (denoted RetGK1 in Table 2) can be quite long, the method can be efficiently approximated (RetGK11 in Table 2) while preserving good accuracy scores. FGSD [30] is a finite-dimensional graph embedding that does not leverage attributes, and reaches state-of-the-art results on different datasets. Finally, [31] is a neural network approach that also reaches top-tier results. One could also compare our results on the REDDIT datasets to the ones of [16], where authors also use persistence diagrams to feed a network (using as first channel a particular case of PersLay, see Section 3), achieving 54.5% and 44.5% of accuracy on REDDIT5K and REDDIT12K respectively.

In order to extract topological features, we use this general scheme: for each graph we compute the HKS filtrations at one or two time step (e.g. t with values in $\{0.1, 1, 10, 100\}$). After generating the corresponding extended persistence diagram $\mathrm{Dg}(\mathrm{hks}_t)$ induced by HKS (see §2 and §4.2), we eventually keep, for each diagram, the first k points which are the farthest away from the diagonal (a specific k is fixed for each dataset, see Table 5 in the supplementary material). We combine these topological features with more traditional graph features formed by the eigenvalues of the normalized graph Laplacian along with the deciles of the computed HKS (right-side channel in Figure 3). So as to evaluate the impact of persistence diagrams, we also report classification performances for those features alone (thus unplugging the PersLay layers) denoted "Spectral + HKS"[5] in Table 2.

We use the same accuracy metric as in [34]. For each dataset, we compute a final score by averaging ten 10-folds, 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 report in Table 2 the average and standard deviation of the scores we obtain. In most cases, our approach is comparable, if not better, than state-of-the-art results, despite using a very simple neural network architecture. More importantly, it can be observed from the last two columns of Table 2 that, for all datasets, the use of extended persistence diagrams significantly improves over using the additional features alone.

5 Conclusion

In this article, we propose a versatile, powerful and simple neural network layer to process persistence diagrams called PersLay, which generalizes most of the techniques used to vectorize persistence diagrams that can be found in the literature—while optimizing them task-wise. Our code is freely available publicly at `https://github.com/MathieuCarriere/perslay`. We showcase the efficiency of our approach by achieving state-of-the-art results on synthetic orbit classification coming from dynamical systems and several graph classification problems from real-life data, while working at larger scales than kernel methods developed for persistence diagrams and remaining simpler than most of its neural network competitors. We believe that PersLay has the potential to become a central tool to incorporate topological descriptors in a wide variety of complex machine learning tasks.

References

[1] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `http://tensorflow.org/`. Software available from tensorflow.org.

[2] Adams, H., Emerson, T., Kirby, M., Neville, R., Peterson, C., Shipman, P., Chepushtanova, S., Hanson, E., Motta, F., and Ziegelmeier, L. Persistence images: a stable vector representation of persistent homology. *Journal of Machine Learning Research*, 18(8), 2017.

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

[4] Buchet, M., Hiraoka, Y., and Obayashi, I. Persistent homology and materials informatics. In *Nanoinformatics*, pp. 75–95. 2018.

[5] Cámara, P. Topological methods for genomics: present and future directions. *Current Opinion in Systems Biology*, 1:95–101, feb 2017.

[6] Carrière, M. `sklearn-tda`: a scikit-learn compatible python package for Machine Learning and TDA. `https://github.com/MathieuCarriere/sklearn_tda`, 2018.

[7] Carrière, M. and Bauer, U. On the metric distortion of embedding persistence diagrams into separable Hilbert spaces. *Accepted for publication in International Symposium on Computational Geometry*, 2019.

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

[9] Carrière, M., Cuturi, M., and Oudot, S. Sliced Wasserstein kernel for persistence diagrams. In *International Conference on Machine Learning*, volume 70, pp. 664–673, jul 2017.

[10] Chazal, F., de Silva, V., and Oudot, S. Persistence stability for geometric complexes. *Geometriae Dedicata*, 173(1):193–214, 2014.

[11] Chazal, F., Fasy, B. T., Lecci, F., Rinaldo, A., and Wasserman, L. Stochastic convergence of persistence landscapes and silhouettes. *Journal of Computational Geometry*, 6(2):140–161, 2015.

[12] Chazal, F., de Silva, V., Glisse, M., and Oudot, S. *The structure and stability of persistence modules*. Springer International Publishing, 2016.

[13] Cohen-Steiner, D., Edelsbrunner, H., and Harer, J. Extending persistence using Poincaré and Lefschetz duality. *Foundations of Computational Mathematics*, 9(1):79–103, feb 2009.

[14] Edelsbrunner, H. and Harer, J. *Computational topology: an introduction*. American Mathematical Society, 2010.

[15] Hertzsch, J.-M., Sturman, R., and Wiggins, S. Dna microarrays: design principles for maximizing ergodic, chaotic mixing. *Small*, 3(2):202–218, 2007.

[16] Hofer, C., Kwitt, R., Niethammer, M., and Uhl, A. Deep learning with topological signatures. In *Advances in Neural Information Processing Systems*, pp. 1634–1644, 2017.

[17] Hu, N., Rustamov, R., and Guibas, L. Stable and informative spectral signatures for graph matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2305–2312, 2014.

[18] Kališnik, S. Tropical coordinates on the space of persistence barcodes. *Foundations of Computational Mathematics*, pp. 1–29, jan 2018.

[19] Kingma, D. and Ba, J. Adam: a method for stochastic optimization. *arXiv*, dec 2014.

[20] Kusano, G., Hiraoka, Y., and Fukumizu, K. Persistence weighted Gaussian kernel for topological data analysis. In *International Conference on Machine Learning*, volume 48, pp. 2004–2013, jun 2016.

[21] Le, T. and Yamada, M. Persistence Fisher kernel: a Riemannian manifold kernel for persistence diagrams. In *Advances in Neural Information Processing Systems*, pp. 10027–10038, 2018.

[22] Li, C., Ovsjanikov, M., and Chazal, F. Persistence-based structural recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2003–2010, jun 2014.

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

[24] Perea, J. and Harer, J. Sliding windows and persistence: an application of topological methods to signal analysis. *Foundations of Computational Mathematics*, 15(3):799–838, jun 2015.

[25] Reininghaus, J., Huber, S., Bauer, U., and Kwitt, R. A stable multi-scale kernel for topological machine learning. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[26] Sun, J., Ovsjanikov, M., and Guibas, L. A concise and provably informative multi-scale signature based on heat diffusion. *Computer graphics forum*, 28:1383–1392, 2009.

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

[28] Tran, Q. H., Vo, V. T., and Hasegawa, Y. Scale-variant topological information for characterizing complex networks. *arXiv preprint arXiv:1811.03573*, 2018.

[29] Tsitsulin, A., Mottin, D., Karras, P., Bronstein, A., and Müller, E. Netlsd: hearing the shape of a graph. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2347–2356. ACM, 2018.

[30] Verma, S. and Zhang, Z.-L. Hunt for the unique, stable, sparse and fast feature learning on graphs. In *Advances in Neural Information Processing Systems*, pp. 88–98, 2017.

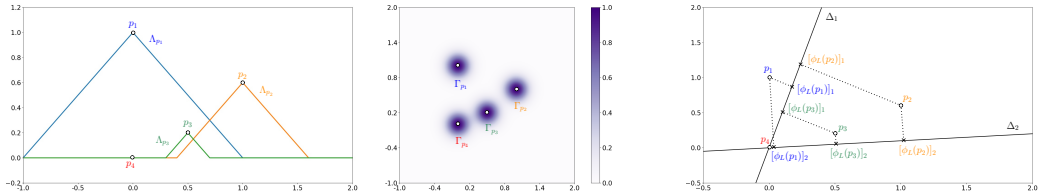
[31] Xinyi, Z. and Chen, L. Capsule graph neural network. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=Byl8BnRcYm`.

[32] Yanardag, P. and Vishwanathan, S. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pp. 1365–1374, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3664-2. doi: 10.1145/2783258.2783417. URL `http://doi.acm.org/10.1145/2783258.2783417`.

[33] Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R., and Smola, A. Deep sets. In *Advances in Neural Information Processing Systems*, pp. 3391–3401, 2017.

[34] Zhang, Z., Wang, M., Xiang, Y., Huang, Y., and Nehorai, A. RetGK: Graph Kernels based on Return Probabilities of Random Walks. In *Advances in Neural Information Processing Systems*, pp. 3968–3978, 2018.

A Diagram vectorization techniques



(a) 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.

(b) Example of Gaussian functions on a persistence diagram with four points p_1, p_2, p_3, p_4.

(c) 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.

Figure 4: Illustration of commonly used diagram vectorizations that are particular cases of PersLay.

B Datasets description

Tables 3 and 4 summarizes key information for each dataset for both our experiments. We also provide an illustration of the orbit we generated (§4.1).

Dataset	Nb of orbit observed	Number of classes	Number of points per orbit
`ORBIT5K`	5000	5	1000
`ORBIT100K`	100000	5	1000

Table 3: Description of the two orbits dataset we generated. The five classes correspond to the five parameter choices for $r \in \{2.5, 3.5, 4.0, 4.1, 4.3\}$. In both `ORBIT5K` and `ORBIT100K`, classes are balanced.

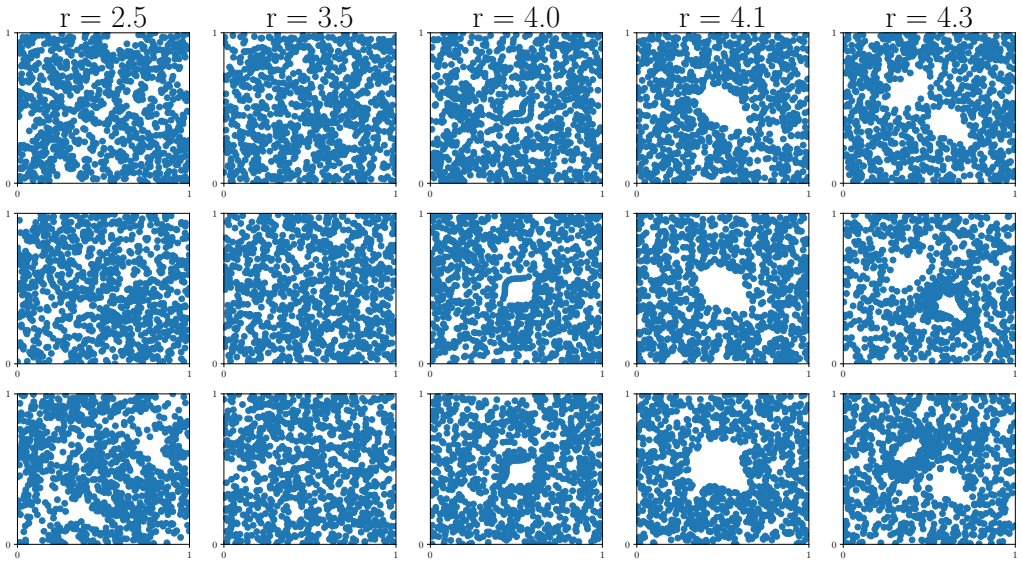


Figure 5: Some example of orbits generated by the different choices of r (three simulations are represented for the different values of r).

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-M	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 4: 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 $\beta_1 = \#\{\text{edges}\} - \#\{\text{nodes}\}$.

Dataset	Func. used	PD preproc.	DeepSet Channel	Optim.
ORBIT5K	Alpha_0, Alpha_1	prom(400)	Pm(25,25,10,top-5)	adam(0.01, 0., 300)
ORBIT100K	Alpha_0, Alpha_1	prom(400)	Pm(25,25,10,top-5)	adam(0.01, 0., 300)
REDDIT5K	$\text{hks}_{1.0}$	prom(300)	Pm(25,25,10,sum)	adam(0.01, 0.99, 500)
REDDIT12K	$\text{hks}_{1.0}$	prom(400)	Pm(5,5,10,sum)	adam(0.01, 0.99, 1000)
COLLAB	$\text{hks}_{0.1}$, hks_{10}	prom(200)	Pm(5,5,10,sum)	adam(0.01, 0.9, 1000)
IMDB-B	$\text{hks}_{0.1}$, hks_{10}	prom(200)	Im(20,(10,2),20,sum)	adam(0.01, 0.9, 500)
IMDB-M	$\text{hks}_{0.1}$, hks_{10}	prom(500)	Im(10,(10,2),10,sum)	adam(0.01, 0.9, 500)
BZR	$\text{hks}_{0.1}$, hks_{10}	—	Im(15,(10,2),10,sum)	adam(0.01, 0.9, 100)
COX2	$\text{hks}_{0.1}$, hks_{10}	—	Im(20,(10,2),20,sum)	adam(0.01, 0.9, 500)
DHFR	$\text{hks}_{0.1}$, hks_{10}	—	Im(20,(10,2),20,sum)	adam(0.01, 0.9, 500)
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, 70)
NCI1	$\text{hks}_{0.1}$, hks_{10}	—	Pm(25,25,10,sum)	adam(0.01, 0.9, 300)
NCI109	$\text{hks}_{0.1}$, hks_{10}	—	Pm(25,25,10,sum)	adam(0.01, 0.9, 300)
FRANKENSTEIN	hks_{10}	—	Im(20,(10,2),20,sum)	adam(0.01, 0.9, 300)

Table 5: Settings used to generate our experimental results.

C Parameters used in our experiments

Input data was fed to the network with mini-batches of size 128. For each dataset, various parameters are given (extended persistence diagrams, neural network architecture, optimizers, etc.) that were used to obtain the scores from Table 2. In Table 5, we use the following shortcuts:

- Alpha_d: persistence diagrams obtained with `Gudhi`'s d-dimensional `AlphaComplex` filtration.
- hks_t: extended persistence diagram obtained with HKS on the graph with parameter t.
- prom(k): preprocessing step selecting the k points that are the farthest away from the diagonal.
- PersLay channel 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`.
- PersLay channel 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 [33] in dimension d_2, 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 [19] with learning rate λ, using an Exponential Moving Average[2] with decay rate d, and run during e epochs.

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

D Proof of Theorem 4.2

The proof directly follows from the following two theorems. This first one, proved in [17], is a consequence of classical arguments from matrix perturbation theory.

Theorem D.1 ([17], Theorem 1) *Let $t \geq 0$ and let L_w be the Laplacian matrix of a graph G with n vertices. Let $\lambda_1 < \cdots < \lambda_k$, $k \leq n$ be the distinct eigenvalues of L_w and denote by $\delta > 0$ the smallest distance between two distinct eigenvalues: $\delta = \min_{j=1,\cdots,k-1} |\lambda_{j+1} - \lambda_j|$. Let G' be another graph with n vertices and Laplacian matrix $\tilde{L}_w = L_w + W$ with $\|W\| < \delta$, where $\|W\|$ denotes the Frobenius norm of W. Then, if $k = n$, there exists a constant $C_0(G,t) > 0$ such that for any vertex $v \in G$,*

$$|\mathrm{hks}_{G,t}(v) - \mathrm{hks}_{G',t}(v)| \leq C_0(G,t)\|W\|;$$

if $k < n$, there exists two constants $C_1(G,t), C_2(G,t) > 0$ such that for any vertex $v \in G$,

$$|\mathrm{hks}_{G,t}(v) - \mathrm{hks}_{G',t}(v)| \leq C_1(G,t)\frac{\|W\|}{\delta - \|W\|} + C_2(G,t)\|W\|$$

In particular, if $\|W\| < \frac{\delta}{2}$, there exists a constant $C(G,t) > 0$ - notice that δ also depends on G - such that in the two above cases,

$$|\mathrm{hks}_{G,t}(v) - \mathrm{hks}_{G',t}(v)| \leq C(G,t)\|W\|.$$

Theorem 4.2 then immediately follows from the second following theorem, which is a special case of general stability results for persistence diagrams.

Theorem D.2 ([12, 13]) *Let $G = (V, E)$ be a graph and $f, g : V \to \mathbb{R}$ be two functions defined on its vertices. Then:*

$$\mathrm{d}_B(\mathrm{Dg}(G,f), \mathrm{Dg}(G,g)) \leq \|f - g\|_\infty, \tag{4}$$

where d_B stands for the so-called bottleneck distance *between persistence diagrams and $\|f - g\|_\infty = \sup_{v \in G} |f(v) - g(v)|$. Moreover, this inequality is also satisfied for each of the subtypes $\mathrm{Ord}_0, \mathrm{Rel}_1, \mathrm{Ext}_0^+$ and Ext_1^- individually.*