PersLay: A Neural Network Layer for Persistence Diagrams and New Graph Topological Signatures

Mathieu Carrière
Columbia University

Frédéric Chazal
Inria Saclay

Yuichi Ike
Fujitsu Laboratories

Théo Lacombe
Inria Saclay

Martin Royer
Inria Saclay

Yuhei Umeda
Fujitsu Laboratories

Abstract

Persistence diagrams, the most common descriptors of Topological Data Analysis, encode topological properties of data and have already proved pivotal in many different applications of data science. However, since the metric space of persistence diagrams is not Hilbert, they end up being difficult inputs 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 space or implicit infinite dimensional Hilbert space with kernels.

In this work, we focus on persistence diagrams built on top of graphs. Relying on extended persistence theory and the so-called heat kernel signature, we show how graphs can be encoded by (extended) persistence diagrams in a provably stable way. We then propose a general and versatile framework for learning vectorizations of persistence diagrams, which encompasses most of the vectorization techniques used in the literature. We finally showcase the experimental strength of our setup by achieving competitive scores on classification tasks on real-life graph datasets.

1 INTRODUCTION

Topological Data Analysis (TDA) is a field of data science whose goal is to detect and encode topological features (such as 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, with its coordinates encoding the feature size. This descriptor has been successfully used in many different applications of data science, such as signal analysis (Perea & Harer, 2015), material science (Buchet et al., 2018), cellular data (Cámara, 2017), or shape recognition (Li et al., 2014) to name a few. This wide range of applications is mainly due to the fact that persistence diagrams encode information based on topology, and as such this information is very often complementary to the one retrieved 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 addition and scalar multiplication, which unfortunately dramatically impedes their use in machine learning applications. To handle this issue, a lot of attention has been devoted to *vectorizations* of persistence diagrams through the construction of either *finite-dimensional embeddings* (Adams et al., 2017; Carrière et al., 2015; Chazal et al., 2015; Kališnik, 2018), i.e., embeddings turning persistence diagrams into vectors in Euclidean space $\mathbb{R}^d$, or *kernels* (Bubenik, 2015; Carrière et al., 2017; Kusano et al., 2016; Le & Yamada, 2018; Reininghaus et al., 2015), i.e., generalized scalar products that implicitly turn persistence diagrams into elements of infinite-dimensional Hilbert spaces.

Even though these methods improved the use of persistence diagrams in machine learning tremendously, several issues remain. For instance, most of these vectorizations only have a few trainable parameters, which may prevent them from fitting well to specific appli-

Proceedings of the 23[rd]International Conference on Artificial Intelligence and Statistics (AISTATS) 2020, Palermo, Italy. PMLR: Volume 108.

cations. As a consequence, it may be very difficult to determine which vectorization is going to work best for a given task. Furthermore, kernel methods (which are generally efficient in practice) 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.

In this work, we show how to use neural networks for handling persistence diagrams. Contrary to static vectorization methods proposed in the literature, we actually learn the vectorization with respect to the learning task that is being solved. Moreover, our framework is general enough so that most of the common vectorizations of the literature (Adams et al., 2017; Bubenik, 2015; Chazal et al., 2015) can be retrieved from our method by specifying its parameters accordingly.

1.1 Our contributions

The contribution of this paper is two-fold.

First, we introduce in Section 2 a new family of topological signatures on graphs: the *extended* persistence diagrams built from the *Heat Kernel Signatures* (HKS) of the graph. These signatures depend on a diffusion parameter $t \geqslant 0$. Although HKS are well-known signatures, they have never been used in the context of persistent homology to encode topological information for graphs. We prove that the resulting diagrams are stable with respect to both the input graph and the parameter t. The use of extended persistence, by opposition to the commonly used *ordinary* persistence, is introduced in order to handle "essential" components, see Section 2.1 below. To our knowledge, it is the first use of extended persistence in a machine learning context. We also experimentally showcase its strength over ordinary persistence on several applications.

Second, building on the recent introduction of *Deep Sets* from (Zaheer et al., 2017), we apply and extend that framework for persistence diagrams, implementing PersLay: a simple, highly versatile, automatically differentiable layer for neural network architectures that can process topological information encoded in persistence diagrams computed from all sorts of datasets. Our framework encompasses most of the common vectorizations that exist in the literature, and we give a necessary and sufficient condition for the learned vectorization to be continuous, improving on the analysis of (Hofer et al., 2019). Using a large-scale dataset coming from dynamical systems which is commonly used in the TDA literature, we give in Section 3.2 a proof-of-concept of the scalability and efficiency of this neural network approach over standard methods in TDA. The implementation of PersLay is publicly available[1] as a plug-and-play Python package based on `tensorflow`, as well as a module of the `Gudhi`[2] library.

We finally combine these two contributions in Section 4 by performing real-graph classification application with benchmark datasets coming various fields of science, such as biology, chemistry and social sciences.

1.2 Related work

Various techniques have been proposed to encode the topological information that is contained in the structure of a given graph, see for instance (Archambault et al., 2007; Li et al., 2012; Ferrara & Fiumara, 2012). In this work, we focus on topological features computed with persistent homology (see Section 1.3 below). This requires to define a real-valued function f on the nodes of the graph. A first simple choice—made in (Hofer et al., 2017)—is to map each node to its degree. In another recent work (Zhao & Wang, 2019), authors proposed to use the Jaccard index and the Ricci curvature. In (Tran et al., 2018), authors adopt a slightly different approach: given a graph with N nodes indexed by $\{1 \dots N\}$ and an integer parameter τ, they compute an $N \times N$ matrix $(p_\tau(i|j))_{i,j}$ where $p_\tau(i|j)$ is the probability that a random walk starting at node j ends at node i after τ steps. This matrix can be thought of as a finite metric space, i.e., a set of N points embedded in $\mathbb{R}^N$, on which topological descriptors can also be computed (Chazal et al., 2014). Here, τ acts as a scale parameter: small values of τ will encode local information while large values will catch large-scale features. Our approach, presented in Section 2, shares the same scale-parametric idea, with the notable difference that we compute our topological descriptors on the graphs directly.

The first approach to feed a neural network architecture with a persistence diagram was presented in (Hofer et al., 2017). It amounts to evaluating the points of the persistence diagram against one (or more) Gaussian distributions with parameters (μ, σ) that are learned during the training process (see Section 3 for more details). Such a transformation is oblivious to any ordering of the diagram points, which is a suitable property, and is a particular case of *permutation invariant* transformations. These general transformations are studied in (Zaheer et al., 2017), and used to define neural networks on sets. These networks are referred to as *Deep Sets*. In particular, the authors in (Zaheer et al., 2017) observed that any permutation invariant

[1] `https://github.com/MathieuCarriere/perslay`
[2] `http://gudhi.gforge.inria.fr/python/latest/`

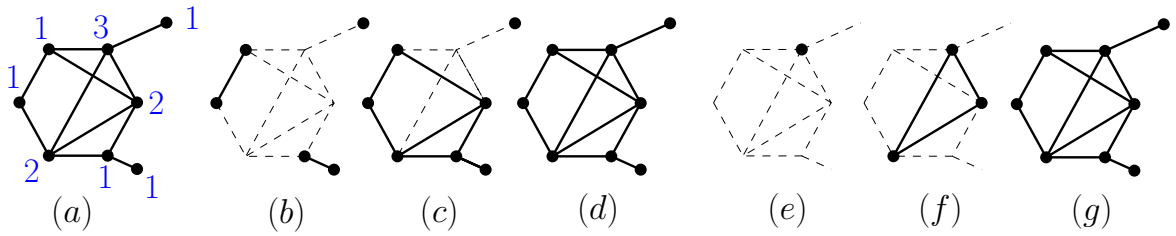

Figure 1: Illustration of sublevel and superlevel graphs. (a) Input graph (V, E) along with the values of a function $f : V \to \mathbb{R}$ (blue). (b, c, d) Sublevel graphs for $\alpha = 1, 2, 3$ respectively. (e, f, g) Superlevel graphs for $\alpha = 3, 2, 1$ respectively.

function L defined on point clouds supported on $\mathbb{R}^p$ with exactly n points can be written of the form:

$$L(\{x_1 \dots x_n\}) = \rho\left(\sum_{i=1}^{n} \phi(x_i)\right), \tag{1}$$

for some $\phi : \mathbb{R}^p \to \mathbb{R}^q$ and $\rho : \mathbb{R}^q \to \mathbb{R}^q$. Obviously, the converse is true: any function defined by (1) is a permutation invariant function. Hofer et al. make use of this idea in (Hofer et al., 2019), where they suggest three possible functions $\phi : \mathbb{R}^2 \to \mathbb{R}$, which roughly correspond to Gaussian, spike and cone functions centered on the diagram points. Our framework builds on the same idea, but substantially generalize theirs, as we are able to generate many more possible vectorizations and ways to combine them (for instance by using a maximum instead of a sum in Equation 1). We deepen the analysis by observing how common vectorizations of the TDA literature can be obtained again as specific instances of our architecture (Section 3). Moreover, we allow for more general weight functions that provide additional interpretability, as shown in Supplementary Material, Section C.

1.3 Background on ordinary persistence

In this section, we briefly recall the basics of ordinary persistence theory. We refer the interested reader to (Cohen-Steiner et al., 2009; Edelsbrunner & Harer, 2010; Oudot, 2015) for a thorough description.

Let X be a topological space, and $f\colon X \to \mathbb{R}$ be a real-valued continuous function. The *α-sublevel set* of X is then defined as: $X_\alpha = \{x \in X \,:\, f(x) \leq \alpha\}$. Making α increase from $-\infty$ to $+\infty$ gives an increasing sequence of sublevel sets, called the *filtration* induced by f. It starts with the empty set and ends with the whole space X (see $(b-d)$ in Figure 1 for an illustration on a graph). Ordinary persistence keeps track of the times of appearance and disappearance of topological features (connected components, loops, cavities, etc.) in this sequence. For instance, one can store the value α_b, called the *birth time*, for which a new connected component appears in X_{α_b}. This connected component eventually gets merged with another one for some value $\alpha_d \geq \alpha_b$, which is stored as well and called the *death time*. Moreover, one says that the component *persists* on the corresponding 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 $\{(\alpha_b, \alpha_d)\}$.

The space of persistence diagrams can be equipped with a parametrized metric d_s, $1 \leqslant s \leqslant \infty$, whose proper definition is not required in this work and is given in Supplementary Material, Appendix A for the sake of completeness. In the particular case $s = \infty$, this metric will be refered to as the *bottleneck* distance between persistence diagrams.

2 EXTENDED PERSISTENCE DIAGRAMS

2.1 Extended persistence

In general ordinary persistence does not fully encode the topology of 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 $\alpha \in \mathbb{R}$, $V_\alpha = \{v \in V \,:\, f(v) \leqslant \alpha\}$, 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 $(G_\alpha)_\alpha$, the 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.

To handle this issue, extended persistence refines the analysis by also looking at the so-called *superlevel set* $X^\alpha = \{x \in X \,:\, f(x) \geqslant \alpha\}$. Similarly to ordinary

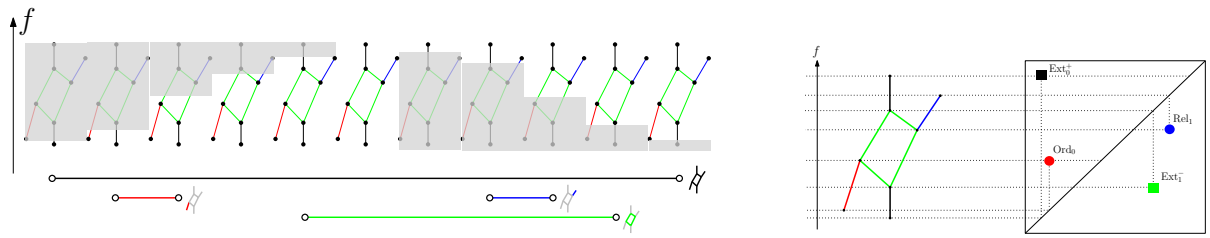

Figure 2: Extended persistence diagram computed on a graph: topological features of the graph are detected in the sequence of sublevel and superlevel graphs shown on the left of the figure. The corresponding intervals are displayed under the sequence: the black interval represents the connected component 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.

persistence on sublevel sets, making α decrease from $+\infty$ to $-\infty$ produces 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, 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.

While it encodes more information than ordinary persistence, extended persistence ensures that points have finite coordinates. In comparison, methods relying on ordinary persistence have to design specific tools to handle points with infinite coordinates (Hofer et al., 2017, 2019), or simply ignore them (Carrière et al., 2017), losing information in the process. Therefore extended persistence allows the use of generic architectures regardless of the homology dimension. Empirical performances show substantial improvement over using ordinary persistence only (see Supplementary Material, Table 7). In practice, computing extended persistence diagrams can be efficiently done with the C++/Python `Gudhi` library (The GUDHI Project, 2015). Persistence diagrams are usually compared with the so-called bottleneck distance d_B—whose proper definition is not required for this work and is recalled in Supplementary Material, Section A. However, the resulting metric space is not Hilbert and as such, incorporating diagrams in a learning pipeline requires to design specific tools, which we do in Section 3.

We recall that extended persistence diagrams can be computed only after having defined a real-valued function on the nodes of the graphs. In the next section, we define a family of such functions from the so-called Heat Kernel Signatures (HKS) for graphs, and show that these signatures enjoy stability properties. Moreover, Section 4 will further demonstrate that they lead to competitive results for graph classification.

2.2 Heat kernel signatures

HKS is an example of spectral family of signatures, that is, functions derived from the spectral decomposition of graph Laplacians, which provide informative features for graph analysis. We start this section with a few basic definitions. 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 $0 \leqslant \lambda_1 \leqslant \cdots \leqslant \lambda_n \leqslant 2$. 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* (HKS):

Definition 2.1 ((Hu et al., 2014; Sun et al., 2009)). *Given a graph G and $t \geq 0$, the* Heat Kernel Signature *with diffusion parameter 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 (Hu et al., 2014) or to define spectral descriptors to compare graphs (Tsitsulin et al., 2018). 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. For the sake of concision, we denote by $\mathrm{Dg}(G,t)$ the extended persistence diagram obtained from a graph G using the filtration induced by the HKS with diffusion parameter t, that is $\mathrm{Dg}(G, \mathrm{hks}_{G,t})$. The following theorem shows these diagrams to be stable with respect to the bottleneck distance d_B between persistence diagrams. The proof can be found in Supplementary Material, Section A.

Theorem 2.2. Stability w.r.t. graph perturbations. *Let* $t \geqslant 0$ *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* $\|W\|_F$, *where* $\|\cdot\|_F$ *denotes the Frobenius norm:*

$$\mathrm{d}_B(\mathrm{Dg}(G,t), \mathrm{Dg}(G',t)) \leqslant C(G,t)\|W\|_F. \qquad (2)$$

On the influence of diffusion parameter t. Building an extended persistence diagram on top of a graph G with the Heat Kernel Signatures requires to pick a specific value of t. In particular, understanding the influence and thus the choice of the diffusion parameter t is an important question for statistical and learning applications. First, we state that the map $t \mapsto \mathrm{Dg}(G,t)$ is Lipschitz-continuous. The proof is found in Supplementary Material, Section A.

Theorem 2.3. Stability w.r.t. parameter t. *Let* G *be a graph. The map* $t \mapsto \mathrm{Dg}(G,t)$ *is* 2*-Lipschitz continuous, that is for* $t, t' \in \mathbb{R}$,

$$\mathrm{d}_B(\mathrm{Dg}(G,t), \mathrm{Dg}(G,t')) \leqslant 2|t - t'| \qquad (3)$$

It follows from Theorem 2.3 that persistence diagrams are robust to the choice of t. An empirical illustration is shown in the supplementary material, Figures 7 and 8.

3 NEURAL NETWORK LEARNING WITH PERSLAY

In this section, we introduce PersLay: a general and versatile neural network layer for learning persistence diagram vectorizations.

3.1 PersLay

In order to define a layer for persistence diagrams, we modify the Deep Set architecture (Zaheer et al., 2017) by defining and implementing a series of new permutation invariant layers, so as to be able to recover and generalize standard vectorization methods used in Topological Data Analysis. To that end we define our generic neural network layer for persistence diagrams, that we call PersLay, through the following equation:

$$\textsc{PersLay}(\mathrm{Dg}) := \mathtt{op}\left(\{w(p)\cdot\phi(p)\}_{p\in\mathrm{Dg}}\right), \qquad (4)$$

where $\mathtt{op}$ 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 representation function that we call *point transformation*, mapping each point (α_b, α_d) of a persistence diagram to a vector.

In practice, w and ϕ are of the form $w_{\theta_1}, \phi_{\theta_2}$ where the gradients of $\theta_1 \mapsto w_{\theta_1}$ and $\theta_2 \mapsto \phi_{\theta_2}$ are known and implemented so that back-propagation can be performed, and the parameters θ_1, θ_2 can be optimized during the training process. 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 three point transformation functions that we use and implement for parameter ϕ in Equation (4).

- The *triangle point transformation* $\phi_\Lambda : \mathbb{R}^2 \to \mathbb{R}^q, p \mapsto \left[\Lambda_p(t_1), \Lambda_p(t_2), \ldots, \Lambda_p(t_q)\right]^T$ 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* $\phi_\Gamma : \mathbb{R}^2 \to \mathbb{R}^q, p \mapsto \left[\Gamma_p(t_1), \Gamma_p(t_2), \ldots, \Gamma_p(t_q)\right]^T$, 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 $\Delta_1, \ldots, \Delta_q$ are q lines in the plane.

Formulation (4) is very general: despite its simplicity, it allows to remarkably encode most of classical persistence diagram vectorizations 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 it connects to most of the popular vectorizations and kernel methods for persistence diagrams in the literature.

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

Table 1: Performance table. PSS-K, PWG-K, SW-K, PF-K stand for *Persistence Scale Space Kernel* (Reininghaus et al., 2015), *Persistence Weighted Gaussian Kernel* (Kusano et al., 2016), *Sliced Wasserstein Kernel* (Carrière et al., 2017) and *Persistence Fisher Kernel* (Le & Yamada, 2018) respectively. We report the scores given in (Le & Yamada, 2018) for competitors on ORBIT5K, and the one we obtained using PersLay for both the ORBIT5K and ORBIT100K datasets.

- Using: $\phi = \phi_\Lambda$ with samples $t_1, \dots, t_q \in \mathbb{R}$, op $= k$th largest value, $w = 1$ (a constant weight function), amounts to evaluating the kth *persistence landscape* (Bubenik, 2015) on $t_1, \dots, t_q \in \mathbb{R}$.
- Using $\phi = \phi_\Lambda$ with samples $t_1, \dots, t_q \in \mathbb{R}$, op $=$ sum, arbitrary weight function w, amounts to evaluating the *persistence silhouette* weighted by w (Chazal et al., 2015) on $t_1, \dots, t_q \in \mathbb{R}$.
- Using $\phi = \phi_\Gamma$ with samples $t_1, \dots, t_q \in \mathbb{R}^2$, op $=$ sum, arbitrary weight function w, amounts to evaluating the *persistence surface* weighted by w (Adams et al., 2017) 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 (Kusano et al., 2016; Le & Yamada, 2018; Reininghaus et al., 2015).
- 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, weight function $w = 1$, is the approach presented in (Hofer et al., 2017).
- Using $\phi = \phi_L$ with lines $\Delta_1, \dots, \Delta_q \in \mathbb{R}^2$, op $= k$th largest value, weight function $w = 1$, is similar to the approach advocated in (Carrière et al., 2017), 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.

Stability of PersLay. The question of the continuity and stability of persistence diagram vectorizations is of importance for TDA practitioners. In (Hofer et al., 2019, Remark 8), authors observed that the operation defined in (4)—with op=sum—is not continuous in general (with respect to the common persistence diagram metrics). Actually, (Divol & Lacombe, 2019, Prop. 5.1) showed that for all $s \geqslant 1$ the map $(\mathrm{Dg} \mapsto \sum_{p \in \mathrm{Dg}} \phi(p))$ is continuous with respect to the metric d_s if and only if ϕ is of the form $\phi(p) = \varphi(p)\|p - \Delta\|^s$, where $\|p - \Delta\|$ denotes the distance from a point $p \in \mathbb{R}^2$ to the diagonal $\Delta = \{(x, x),\ x \in \mathbb{R}\}$ and φ is a continuous and bounded function. Furthermore, when $s = 1$ and φ is 1-Lipschitz continuous, one can show that the map is actually stable ((Hofer et al., 2019, Thm. 12), (Divol & Lacombe, 2019, Prop. 5.2)), in the following sense:

$$\left\| \sum_{p \in \mathrm{Dg}_1} \phi(p) - \sum_{p' \in \mathrm{Dg}_2} \phi(p') \right\|_\infty \leqslant d_1(\mathrm{Dg}_1, \mathrm{Dg}_2).$$

In particular, this means that requiring continuity for the learned vectorization, as done in (Hofer et al., 2019), implies constraining the weight function to take small values for points close to the diagonal. However, in general there is no specific reason to consider that points close to the diagonal are less important than others, given a learning task.

3.2 A proof of concept: classification on large scale dynamical system dataset

Our first application is on a synthetic dataset used as a benchmark in Topological Data Analysis (Adams et al., 2017; Carrière et al., 2017; Le & Yamada, 2018). It consists in sequences of points generated by different dynamical systems, see (Hertzsch et al., 2007). 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:

$$\begin{cases} x_{n+1} = x_n + r y_n(1 - y_n) & \mod 1 \\ y_{n+1} = y_n + r x_{n+1}(1 - x_{n+1}) & \mod 1 \end{cases} \tag{5}$$

The orbits of this dynamical system heavily depend on parameter r. More precisely, for some values of r, voids might form in these orbits (see Supplementary Material, Figure 4), and as such, persistence diagrams are likely to perform well at attempting to classify orbits with respect to the value of r generating them. As in previous works (Adams et al., 2017; Carrière et al., 2017; Le & Yamada, 2018), 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 $N = 1,000$ points in each simulated orbit. These point clouds are then turned into persistence diagrams using a standard geometric filtration (Chazal et al., 2014), called the AlphaComplex filtration[3] in dimensions 0 and 1. We generate two datasets: The first is ORBIT5K, where for each value of

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

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 (Le & Yamada, 2018). The second is `ORBIT100K`, which 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 large datasets of large diagrams, since all the previous works dealing with this data (Adams et al., 2017; Carrière et al., 2017; Le & Yamada, 2018) use kernel methods.

Results are displayed in Table 1. Not only do we improve on previous results for `ORBIT5K`, we also show with `ORBIT100K` that classification accuracy is further increased as more observations are made available. For consistency we use the same accuracy metric as (Le & Yamada, 2018), that is, we split observations in 70%-30% training-test sets and report the average test accuracy over 100 runs. The parameters used are summarized in Supplementary Material, Section C.

4 APPLICATION TO GRAPH CLASSIFICATION

In order to truly showcase the contribution of PersLay, we use a very simple network architecture, namely a two-layer network. The first layer is PersLay, which processes persistence diagrams. The resulting vector is normalized and fed to the second and final layer, a fully-connected layer whose output is used for predictions. See Figure 3 for an illustration. We emphasize that this simplistic two-layer architecture is designed so as to produce knowledge and understanding (see Supplementary Material, Section C), rather than achieving the best possible performances.

Choice of hyperparameters. In our experiments, we set $w : p \mapsto w_{i,j}\mathbf{1}_{p\in C_{i,j}}$, where $C_{i,j}$ denote the (i,j)-th cell in a $N \times N$ grid discretization of the unit square, and all $(w_{i,j})_{1\leqslant i,j\leqslant N}$ are trainable parameters. N is typically set to 10 or 20. For aggregation operator `op` we use the sum. Further details are given in Supplementary Material, see Table 5 for reporting of the chosen hyperparameters and Table 6 for a study of the influence of the grid size or the choice of ϕ.

Point transformations ϕ are chosen among the three choices $\{\phi_\Lambda, \phi_\Gamma, \phi_L\}$ introduced in Section 3.1. Empirically no representation is uniformly better than the others. The choice of the best point transformation ϕ for a given task could also be selected through a cross-validation scheme, or by learning a linear interpolation between these point transformations: by setting $\phi = \alpha_\Lambda\phi_\Lambda + \alpha_\Gamma\phi_\Gamma + \alpha_L\phi_L$, where $\alpha_\Lambda, \alpha_\Gamma, \alpha_L$ are trainable non-negative weights that sum to 1. Thorough exploration of these alternatives is left for future work.

As mentioned in Section 2, the diagrams we produce are stable with respect to the choice of the HKS diffusion parameter t (Thm. 2.2 and 2.3). As such, we generally use $t = 0.1$ and $t = 10$ in our experiments. We also refer to Supplementary Material where Figure 7 illustrates the evolution of a persistence diagram w.r.t. t and Figure 8 provides the classification accuracy through varying values of t. In practice, it is thus sufficient to sample few values of t using a log-scale, as suggested for example in (Sun et al., 2009, §5).
A subsequent natural question is: given a learning task, can t itself be optimized? The question of optimizing over a family of filtrations induced by parametric functions $\{f_\theta\}_{\theta\in\Theta}$ the map $\theta \mapsto \mathrm{Dg}(G, f_\theta)$ has been studied both theoretically and practically in very recent works (Brüel-Gabrielsson et al., 2019; Leygonie et al., 2019). Hence, we also apply this approach for the filtrations induced by the HKS, optimizing the parameter t during the learning process. Note that the running time of the experiments is greatly increased since one has to recompute all persistence diagrams for each epoch, that is, each time t is updated. Moreover, we noticed after preliminary numerical investigations (see Supplementary Material, Section C) that classification accuracies were not improved by a large margin and remained comparable with results obtained without optimizing t, so we did not include this optimization step in our results.

Table 5 gives a detailed summary report of the different hyper-parameters chosen for each experiment.

Experimental settings. We evaluate our architecture on a series of different graph datasets commonly used as a baseline in graph classification problems. `REDDIT5K, REDDIT12K, COLLAB` (Yanardag & Vishwanathan, 2015), `IMDB-B, IMDB-M` (Tran et al., 2018) are composed of social graphs. `COX2, DHFR, MUTAG, PROTEINS, NCI1, NCI109` are graphs coming from medical or biological frameworks (also from (Tran et al., 2018)). A quantitative summary of these datasets is found in Supplementary Material, Table 4.

We compare performances with five other top graph classification methods. Scale-variant topo (Tran et al., 2018) leverages a kernel for ordinary persistence diagrams computed on point cloud used to encode the graphs. RetGK (Zhang et al., 2018) is a kernel method for graphs that leverages eventual *attributes* on the graph vertices and edges. FGSD (Verma & Zhang, 2017) is a finite-dimensional graph embedding that does not leverage attributes. Finally, GCNN (Xinyi & Chen, 2019) and GIN (Xu et al., 2019) are two graph neural network approaches that reach top-tier results. One could also compare our results on the `REDDIT` datasets to the ones of Hofer et al. (2017), where authors also use persistence diagrams to feed a network

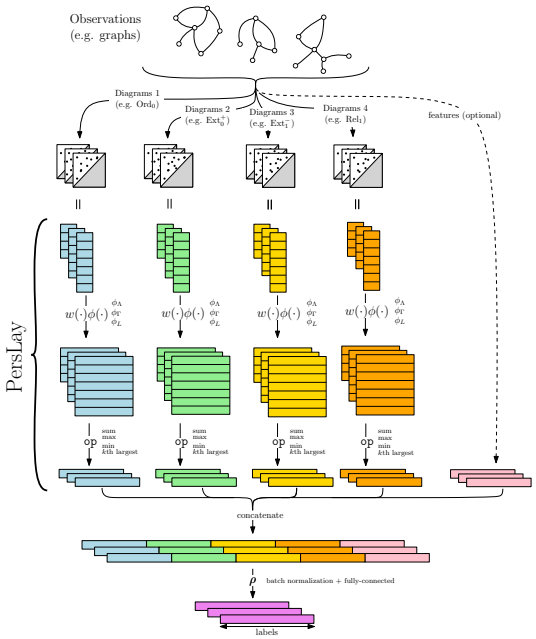

Figure 3: Network architecture illustrated in the case of our graph classification experiments (Section 4). Each graph is encoded as a set of persistence diagrams, then processed by an independent instance of PersLay. Each instance embeds diagrams in some vector space using two functions w, ϕ that are optimized during training and a fixed permutation-invariant operator `op`.

Dataset	SV[1]	RetGK* [2]	FGSD [3]	GCNN [4]	GIN [5]	PersLay Mean	PersLay Max
REDDIT5K	—	56.1	47.8	52.9	57.0	55.6	56.5
REDDIT12K	—	48.7	—	46.6	—	47.7	49.1
COLLAB	—	81.0	80.0	79.6	80.1	76.4	78.0
IMDB-B	72.9	71.9	73.6	73.1	74.3	71.2	72.6
IMDB-M	50.3	47.7	52.4	50.3	52.1	48.8	52.2
COX2*	78.4	80.1	—	—	—	80.9	81.6
DHFR*	78.4	81.5	—	—	—	80.3	80.9
MUTAG*	88.3	90.3	92.1	86.7	89.0	89.8	91.5
PROTEINS*	72.6	75.8	73.4	76.3	75.9	74.8	75.9
NCI1*	71.6	84.5	79.8	78.4	82.7	73.5	74.0
NCI109*	70.5	—	78.8	—	—	69.5	70.1

Table 2: Classification accuracy over benchmark graph datasets. Our results (PersLay, right hand side) are recorded from ten runs of a 10-fold classification evaluation (see Section 4 for details). "Mean" is consistent with (Zhang et al., 2018)[2], while "Max" should be compared to (Tran et al., 2018)[1], (Verma & Zhang, 2017)[3], (Xinyi & Chen, 2019)[4] and (Xu et al., 2019)[5], as it corresponds to the mean accuracy over a *single 10-fold*. The * indicates datasets that contain attributes (labels) on graph nodes and symmetrically the methods that leverage such attributes for classification purposes.

(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.

Topological features were extracted using the graph signatures introduced in Section 2. We combine these diagrams 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). The impact of the topological features in this learning process is evaluated via an ablation study, see Table 7 (Supplementary Material).

For each dataset, we perform 10 ten-fold evaluations and report the average and best ten-fold results. For a ten-fold evaluation, we split the data in 10 equally-sized folds, and record the classification accuracy obtained on the i-th fold (test) after training on the 9 remaining others. Here, i is cycled from 1 to 10. The mean of those 10 ten-fold experiments is naturally more robust for evaluation purposes, and we report it in the column "PersLay - Mean". This is consistent with the evaluation procedure from (Zhang et al., 2018). Simultaneously, we also report the best single 10-fold accuracy obtained, reported in the column "PersLay - Max", which is comparable to the results reported by all the other competitors.

In most cases, our approach is comparable with state-of-the-art results, despite using a very simple neural network architecture. Interestingly, both topology-based methods (SV and PersLay) have mediocre performances on the `NCI` datasets, suggesting that topology is not discriminative for these datasets. Additional experimental results, including ablation studies and variations of hyper-parameters (weight grid size N, diffusion parameter t) are provided in Supplementary Material, Section C.

5 CONCLUSION

In this article, we introduced a new family of topological signatures on graphs, that are both stable and well-formed for learning purposes. In parallel we defined a powerful and versatile 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.

We showcase the efficiency of our approach by achieving state-of-the-art results on synthetic orbit classification coming from dynamical systems and being competitive on 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 based on neural networks. Our code is freely available publicly at `https://github.com/MathieuCarriere/perslay` and is part of the `Gudhi`[4] library.

[4] `http://gudhi.gforge.inria.fr/python/latest/`

References

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.

Archambault, D., Munzner, T., and Auber, D. Topolayout: Multilevel graph layout by topological features. *IEEE transactions on visualization and computer graphics*, 13(2):305–317, 2007.

Brüel-Gabrielsson, R., Nelson, B. J., Dwaraknath, A., Skraba, P., Guibas, L. J., and Carlsson, G. A topology layer for machine learning. *arXiv preprint arXiv:1905.12200*, 2019.

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

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

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

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.

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.

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

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.

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

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.

Divol, V. and Lacombe, T. Understanding the topology and the geometry of the persistence diagram space via optimal partial transport. *arXiv preprint arXiv:1901.03048*, 2019.

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

Ferrara, E. and Fiumara, G. Topological features of online social networks. *arXiv preprint arXiv:1202.0331*, 2012.

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

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.

Hofer, C. D., Kwitt, R., and Niethammer, M. Learning representations of persistence barcodes. *Journal of Machine Learning Research*, 20(126):1–45, 2019. URL `http://jmlr.org/papers/v20/18-358.html`.

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.

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

Kingma, D. and Ba, J. Adam: a method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, dec 2014.

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.

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.

Leygonie, J., Oudot, S., and Tillmann, U. A framework for differential calculus on persistence barcodes, 2019.

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.

Li, G., Semerci, M., Yener, B., and Zaki, M. J. Effective graph classification based on topological and label attributes. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 5(4):265–283, 2012.

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

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.

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.

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.

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

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

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.

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.

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

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? In *ICLR 2019*, 2019.

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`.

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.

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.

Zhao, Q. and Wang, Y. Learning metrics for persistence-based summaries and applications for graph classification. In *Advances in Neural Information Processing Systems 32*, pp. 9859–9870. 2019.