Topological Data Analysis for Machine Learning
Lecture 4: Recent Advances in Topological Machine Learning

Bastian Rieck

Pseudomanifold

Preliminaries

Do you have feedback or any questions? Write to bastian.rieck@bsse.ethz.ch or reach out to @Pseudomanifold on Twitter. You can find the slides and additional information with links to more literature here:

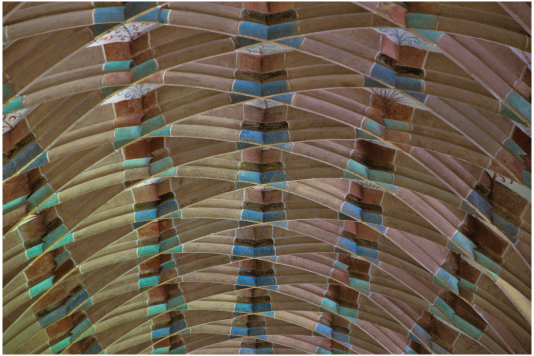
https://topology.rocks/ecml_pkdd_2020

Recap

- The *persistence diagram* is the 'basic' topological feature descriptor.
- Multiple alternatives exist, with different key properties.
- Their choice is application-dependent.

In this lecture

Putting everything together

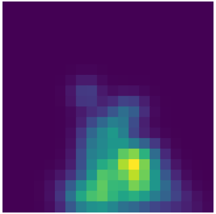


How can we *build* topology-based machine learning models?

Simple feature-based analysis pipeline

Suitable for point clouds, graphs, etc.

1. Pick appropriate filtration
2. Calculate persistence diagrams
3. Vectorise using *persistence images*
4. Use arbitrary feature-based algorithm (SVM, random forest, ...)



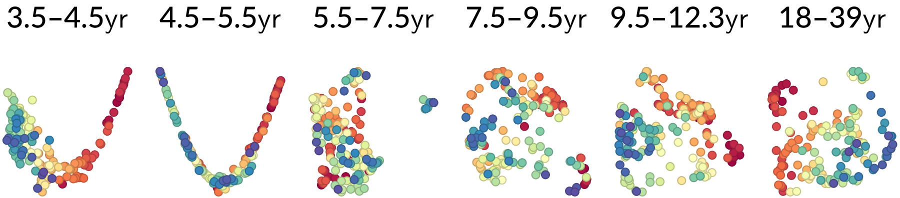
Brief example

B. Rieck et al., 'Uncovering the Topology of Time-Varying fMRI Data using Cubical Persistence', 2020

- Input: fMRI volumes
- Filtration: induced by 'activation function'
- Use persistence images to obtain time-varying embedding
- Describe topological dynamics based on dimensionality reduction algorithm
- Learn about differences of population subgroups

Brief example, continued

Cohort brain trajectories



Classifying *unlabelled* graphs

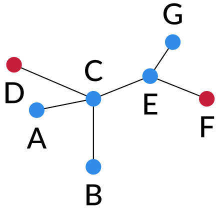
Using 'classical' machine learning models

1. Calculate degree filtration (or another descriptor)
2. Repeat the analysis pipeline described above
3. Learn weights for topological descriptors to improve predictive power[1]

[1] Q. Zhao and Y. Wang, 'Learning metrics for persistence-based summaries and applications for graph classification', *Advances in Neural Information Processing Systems 32 (NeurIPS)*, 2019, pp. 9855–9866

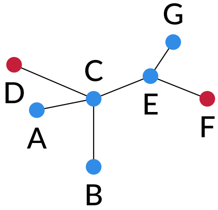
Classifying *labelled* graphs

Weisfeiler–Lehman iteration & subtree feature vector



Classifying *labelled* graphs

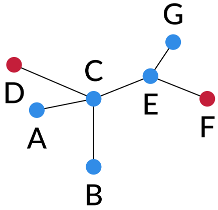
Weisfeiler–Lehman iteration & subtree feature vector



Node	Own label	Adjacent labels
A	🔵	🔵
B	🔵	🔵
C	🔵	🔵 🔵 🔵 🔴
D	🔴	🔵
E	🔵	🔵 🔵 🔴
F	🔴	🔵
G	🔵	🔵

Classifying *labelled* graphs

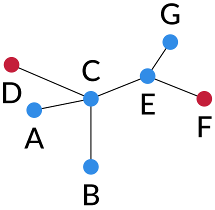
Weisfeiler–Lehman iteration & subtree feature vector



Node	Own label	Adjacent labels	Hashed label
A	●	●	●
B	●	●	●
C	●	● ● ● ●	●
D	●	●	●
E	●	● ● ●	●
F	●	●	●
G	●	●	●

Classifying *labelled* graphs

Weisfeiler–Lehman iteration & subtree feature vector



Label	●	●	●	●
Count	3	1	2	1

$$\Phi(\mathcal{G}) := (3, 1, 2, 1)$$

Compare $\mathcal{G}$ and $\mathcal{G}'$ by evaluating a kernel between $\Phi(\mathcal{G})$ and $\Phi(\mathcal{G}')$ (linear, RBF, ...).

A Persistent Weisfeiler–Lehman Procedure for Graph Classification

A Persistent Weisfeiler–Lehman Procedure for Graph Classification

Bastian Rieck [*1] Christian Bock [*1] Karsten Borgwardt [1]

Abstract

The Weisfeiler–Lehman graph kernel exhibits competitive performance in many graph classification tasks. However, its subtree features are not able to capture connected components and cycles, topological features known for characterising graphs. To extract such features, we leverage propagated node label information and transform unweighted graphs into metric ones. This permits us to augment the subtree features with topological information obtained using *persistent homology*, a concept from topological data analysis. Our method, which we formalise as a generalisation of Weisfeiler–Lehman subtree features, exhibits favourable classification accuracy and its improvements in predictive performance are mainly driven by including cycle information.

1. Introduction

Graph-structured data sets are ubiquitous in a variety of different application domains, each of them posing a separate challenge while also requiring different tasks to be solved. A common task involves *graph classification*, for which a variety of methods exists. These methods comprise convolutional neural networks (Duvenaud et al. 2015), recurrent neural networks (Lei et al. 2017), or Hilbert space methods (Vishwanathan et al. 2010), the latter also being referred to as *graph kernels*. While several approaches for defining graph kernels exist, the most common one uses the $\mathcal{R}$-convolution framework (Haussler 1999), which makes it possible to define the similarity between two graphs as a function of the similarity of their substructures.

Substructures that have been used for graph classification range from graphlets (Shervashidze et al. 2009), i.e. small non-isomorphic graphs of fixed size, over shortest paths (Borgwardt & Kriegel 2005), to random walks (Gärtner et al. 2003, Kashima et al. 2003, Sugiyama & Borgwardt 2015). One of the most powerful substructures is the set of *subtree patterns* (Ramon & Gärtner 2003), i.e. patterns based on rooted subgraphs of a graph. Their computational complexity made their applicability somewhat limited, until the Weisfeiler–Lehman (WL) graph kernel framework (Shervashidze & Borgwardt 2009, Shervashidze et al. 2011) was developed. Properly trained, it still constitutes the state-of-the-art method for many graph classification tasks. The framework is based on the idea of iteratively propagating (node) label information through a graph, leading to a feature vector representation that can be used to assess the dissimilarity of two graphs.

One of the disadvantages of this framework is that its relabelling step, i.e. the step in which subtree patterns are being *compressed*, is somewhat "brittle": labels are only compared with a Dirac kernel, making their dissimilarity a coarse function. Moreover, the subtree feature vector only contains counts of compressed labels and can neither account for their relevance with respect to the topology of the graph nor capture connected components and cycles, both of which are important and interpretable features for characterising graphs (Rieck et al. 2018, Sizemore et al. 2017). We thus propose an enhancement of the original WL stabilisation procedure that uses recent advances in topological data analysis (Munch 2017) to alleviate these issues. Our contributions are as follows:

- We measure the relevance of topological features (connected components and cycles) in graphs and use them to define a novel set of WL subtree features, which we show to be a generalised version of the original ones.
- We develop a topology-based kernel that uses an iterative variant of the WL stabilisation procedure to classify non-attributed graphs.
- We demonstrate that our proposed features perform favourably on a range of graph classification benchmark data sets. In particular, we empirically show that the inclusion of *cycle* information yields classification accuracy improvements over state-of-the-art methods.

[*]Equal contribution [1]Department of Biosystems Science and Engineering, ETH Zurich, 4058 Basel, Switzerland. Correspondence to: Bastian Rieck <bastian.rieck@bsse.ethz.ch>, Karsten Borgwardt <karsten.borgwardt@bsse.ethz.ch>.

Proceedings of the 36th International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019. Copyright 2019 by the author(s).

Christian Bock
chrs_bock

Karsten Borgwardt
kmborgwardt

- The Weisfeiler–Lehman algorithm vectorises labelled graphs
- Persistent homology captures relevant topological features
- We can *combine* them to obtain a *generalised* formulation
- This requires a distance between multisets

A distance between label multisets

Let $A = \{l_1^{a_1}, l_2^{a_2}, \dots\}$ and $B = \{l_1^{b_1}, l_2^{b_2}, \dots\}$ be two multisets that are defined over the same label alphabet $\Sigma = \{l_1, l_2, \dots\}$.

Transform the sets into count vectors, i.e. $\vec{x} := [a_1, a_2, \dots]$ and $\vec{y} := [b_1, b_2, \dots]$.

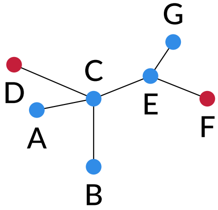
Calculate their *multiset distance* as

$$\operatorname{dist}(\vec{x}, \vec{y}) := \left(\sum_i |a_i - b_i|^p \right)^{\frac{1}{p}},$$

i.e. the p^{th} Minkowski distance, for $p \in \mathbb{R}$. Since nodes and their multisets are in one-to-one correspondence, we now have a metric on the graph!

Multiset distance

Example for $p = 1$



$$
\begin{aligned}
\mathrm{dist}(C, E) &= \mathrm{dist}\left(\{{\color{blue}\bullet}^3, {\color{red}\bullet}^1\}, \{{\color{blue}\bullet}^2, {\color{red}\bullet}^1\}\right) \\
&= \mathrm{dist}([3, 1], [2, 1]) \\
&= 1
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{dist}(C, A) &= \mathrm{dist}\left(\{{\color{blue}\bullet}^3, {\color{red}\bullet}^1\}, \{{\color{blue}\bullet}^1\}\right) \\
&= \mathrm{dist}([3, 1], [1, 0]) \\
&= 3
\end{aligned}
$$

Extending the multiset distance to a distance between vertices

Use vertex label from *previous* Weisfeiler–Lehman iteration, i.e. $\mathrm{l}_{v_i}^{(h-1)}$, as well as $\mathrm{l}_{v_i}^{(h)}$, the one from the *current* iteration:

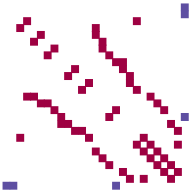
$$\mathrm{dist}(v_i, v_j) := \left[\mathrm{l}_{v_i}^{(h-1)} \neq \mathrm{l}_{v_j}^{(h-1)}\right] + \mathrm{dist}\left(\mathrm{l}_{v_i}^{(h)}, \mathrm{l}_{v_j}^{(h)}\right) + \tau$$

$\tau \in \mathbb{R}_{>0}$ is required to make this into a proper metric. This turns *any* labelled graph into a weighted graph whose persistent homology we can calculate!

Vertex distance, multi-scale properties

Example

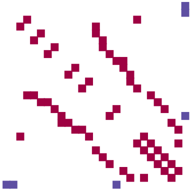
$h = 0$



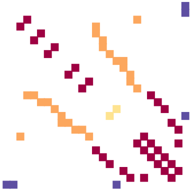
Vertex distance, multi-scale properties

Example

$h = 0$



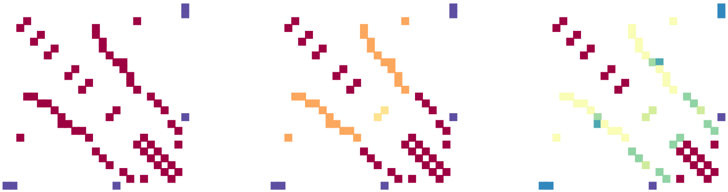
$h = 1$



Vertex distance, multi-scale properties

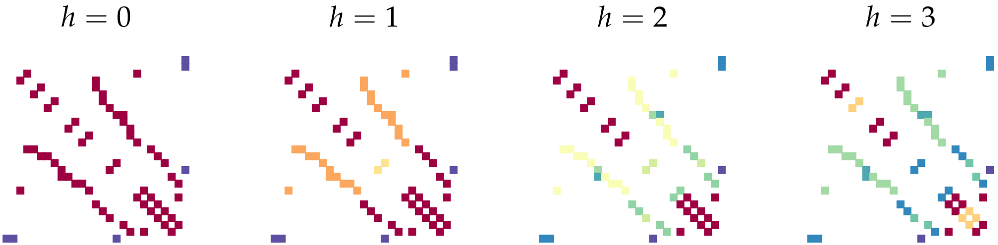
Example

$h = 0$ $h = 1$ $h = 2$



Vertex distance, multi-scale properties

Example



Persistence-based Weisfeiler–Lehman feature vectors

Connected components

$$\Phi^{(h)}_{\text{P-WL}} := \left[\mathfrak{p}^{(h)}\left(l_0\right), \mathfrak{p}^{(h)}\left(l_1\right), \ldots\right]$$

$$\mathfrak{p}^{(h)}\left(l_i\right) := \sum_{\mathrm{l}(v)=l_i} \operatorname{pers}\left(v\right)^p,$$

Cycles

$$\Phi^{(h)}_{\text{P-WL-C}} := \left[\mathfrak{z}^{(h)}\left(l_0\right), \mathfrak{z}^{(h)}\left(l_1\right), \ldots\right]$$

$$\mathfrak{z}^{(h)}\left(l_i\right) := \sum_{l_i \in \mathrm{l}(u,v)} \operatorname{pers}\left(u, v\right)^p,$$

Persistence-based Weisfeiler–Lehman feature vectors

Connected components

$$\Phi^{(h)}_{\text{P-WL}} := \left[\mathfrak{p}^{(h)}\left(l_0\right), \mathfrak{p}^{(h)}\left(l_1\right), \dots\right]$$

$$\mathfrak{p}^{(h)}\left(l_i\right) := \sum_{\mathrm{l}(v)=l_i} \operatorname{pers}\left(v\right)^p,$$

Cycles

$$\Phi^{(h)}_{\text{P-WL-C}} := \left[\mathfrak{z}^{(h)}\left(l_0\right), \mathfrak{z}^{(h)}\left(l_1\right), \dots\right]$$

$$\mathfrak{z}^{(h)}\left(l_i\right) := \sum_{l_i \in \mathrm{l}(u,v)} \operatorname{pers}\left(u, v\right)^p,$$

Bonus

We can re-define the vertex distance to obtain the original Weisfeiler–Lehman subtree features (plus information about cycles):

$$\operatorname{dist}(v_i, v_j) := \begin{cases} 1 & \text{if } v_i \neq v_j \\ 0 & \text{otherwise} \end{cases}$$

Classification results

	D & D	MUTAG	NCI1	NCI109	PROTEINS	PTC-MR	PTC-FR	PTC-MM	PTC-FM
V-Hist	78.32 ± 0.35	85.96 ± 0.27	64.40 ± 0.07	63.25 ± 0.12	72.33 ± 0.32	58.31 ± 0.27	68.13 ± 0.23	66.96 ± 0.51	57.91 ± 0.83
E-Hist	72.90 ± 0.48	85.69 ± 0.46	63.66 ± 0.11	63.27 ± 0.07	72.14 ± 0.39	55.82 ± 0.00	65.53 ± 0.00	61.61 ± 0.00	59.03 ± 0.00
RetGK*	81.60 ± 0.30	90.30 ± 1.10	84.50 ± 0.20		75.80 ± 0.60	62.15 ± 1.60	67.80 ± 1.10	67.90 ± 1.40	63.90 ± 1.30
WL	79.45 ± 0.38	87.26 ± 1.42	85.58 ± 0.15	84.85 ± 0.19	76.11 ± 0.64	63.12 ± 1.44	67.64 ± 0.74	67.28 ± 0.97	64.80 ± 0.85
Deep-WL*		82.94 ± 2.68	80.31 ± 0.46	80.32 ± 0.33	75.68 ± 0.54	60.08 ± 2.55			
P-WL	79.34 ± 0.46	86.10 ± 1.37	85.34 ± 0.14	84.78 ± 0.15	75.31 ± 0.73	63.07 ± 1.68	67.30 ± 1.50	68.40 ± 1.17	64.47 ± 1.84
P-WL-C	78.66 ± 0.32	90.51 ± 1.34	85.46 ± 0.16	84.96 ± 0.34	75.27 ± 0.38	64.02 ± 0.82	67.15 ± 1.09	68.57 ± 1.76	65.78 ± 1.22
P-WL-UC	78.50 ± 0.41	85.17 ± 0.29	85.62 ± 0.27	85.11 ± 0.30	75.86 ± 0.78	63.46 ± 1.58	67.02 ± 1.29	68.01 ± 1.04	65.44 ± 1.18

Try it out

Deep Learning with Topological Signatures

Deep Learning with Topological Signatures

Christoph Hofer
Department of Computer Science
University of Salzburg, Austria
chofer@cosy.sbg.ac.at

Roland Kwitt
Department of Computer Science
University of Salzburg, Austria
Roland.Kwitt@sbg.ac.at

Marc Niethammer
UNC Chapel Hill, NC, USA
mn@cs.unc.edu

Andreas Uhl
Department of Computer Science
University of Salzburg, Austria
uhl@cosy.sbg.ac.at

Abstract

Inferring topological and geometrical information from data can offer an alternative perspective on machine learning problems. Methods from topological data analysis, e.g., persistent homology, enable us to obtain such information, typically in the form of summary representations of topological features. However, such topological signatures often come with an unusual structure (e.g., multisets of intervals) that is highly impractical for most machine learning techniques. While many strategies have been proposed to map these topological signatures into machine learning compatible representations, they suffer from being agnostic to the target learning task. In contrast, we propose a technique that enables us to input topological signatures to deep neural networks and *learn* a task-optimal representation during training. Our approach is realized as a novel input layer with favorable theoretical properties. Classification experiments on 2D object shapes and social network graphs demonstrate the versatility of the approach and, in case of the latter, we even outperform the state-of-the-art by a large margin.

1 Introduction

Methods from algebraic topology have only recently emerged in the machine learning community, most prominently under the term *topological data analysis (TDA)* [7]. Since TDA enables us to infer relevant topological and geometrical information from data, it can offer a novel and potentially beneficial perspective on various machine learning problems. Two compelling benefits of TDA are (1) its versatility, i.e., we are not restricted to any particular kind of data (such as images, sensor measurements, time-series, graphs, etc.) and (2) its robustness to noise. Several works have demonstrated that TDA can be beneficial in a diverse set of problems, such as studying the manifold of natural image patches [8], analyzing activity patterns of the visual cortex [28], classification of 3D surface meshes [27, 22], clustering [11], or recognition of 2D object shapes [29].

Currently, the most widely-used tool from TDA is *persistent homology* [15, 14]. Essentially[1], persistent homology allows us to track topological changes as we analyze data at multiple "scales". As the scale changes, topological features (such as connected components, holes, etc.) appear and disappear. Persistent homology associates a *lifespan* to these features in the form of a *birth* and a *death* time. The collection of (birth, death) tuples forms a multiset that can be visualized as a persistence diagram or a barcode, also referred to as a *topological signature* of the data. However, leveraging these signatures for learning purposes poses considerable challenges, mostly due to their

[1] We will make these concepts more concrete in Sec. 2.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

- Obtain persistence diagrams from graph filtration
- Define layer to *project* persistence diagrams to 1D
- Learn parameters for multiple projections
- Stack projected diagrams and use as features
- First successful combination of deep learning and topology![2]

[2] C. Hofer et al., 'Deep Learning with Topological Signatures', *Advances in Neural Information Processing Systems 30 (NeurIPS)*, Red Hook, NY, USA, 2017, pp. 1634–1644

Details

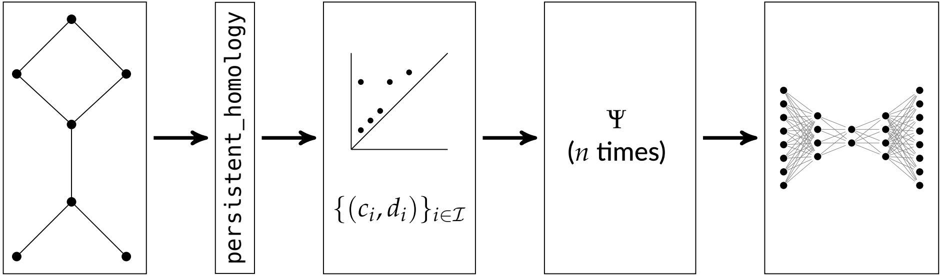
Use a differentiable *coordinatisation* scheme of the form $\Psi\colon \mathcal{D} \to \mathbb{R}$. Letting $p := (c, d)$ for a tuple in a diagram (in creation–persistence coordinates), we have

$$\Psi(p) := \begin{cases} \exp\left(-\sigma_0^2(c-\mu_0)^2 - \sigma_1^2(d-\mu_1)^2\right) & \text{if } c \in [\nu, \infty) \\ \exp\left(-\sigma_0^2(c-\mu_0)^2 - \sigma_1^2(\log(d/\nu)\nu + \nu - \mu_1)^2\right) & \text{if } c \in (0, \nu) \\ 0 & \text{if } c = 0 \end{cases}$$

with $(\mu_0, \mu_1) \in \mathbb{R} \times \mathbb{R}^+$, $(\sigma_0, \sigma_1) \in \mathbb{R}^+ \times \mathbb{R}^+$, and $\nu \in \mathbb{R}^+$ being *trainable* parameters. The whole diagram is then represented as a sum over each individual projections.

Using n different coordinatisations, we obtain a differentiable embedding of a persistence diagram into $\mathbb{R}^n$.

Full classification pipeline



Summary

	REDDIT-5K	REDDIT12K
Graphlet kernel	41.0	31.8
Deep graphlet kernel	41.3	32.2
PATCHY-SAN	49.1	41.3
No essential features	49.1	38.5
With essential features	54.5	44.5

- Excellent performance for social network graph classification.
- Simple to implement and use; feature maps are even interpretable.
- Highly generic & not restricted to graph classification problems.

Try it out

Topological autoencoders

Topological Autoencoders

Michael Moor[†1 2] Max Horn[†1 2] Bastian Rieck[‡1 2] Karsten Borgwardt[‡1 2]

Abstract

We propose a novel approach for preserving topological structures of the input space in latent representations of autoencoders. Using *persistent homology*, a technique from topological data analysis, we calculate topological signatures of both the input and latent space to derive a topological loss term. Under weak theoretical assumptions, we construct this loss in a differentiable manner, such that the encoding learns to retain multi-scale connectivity information. We show that our approach is theoretically well-founded and that it exhibits favourable latent representations on a synthetic manifold as well as on real-world image data sets, while preserving low reconstruction errors.

1. Introduction

While topological features, in particular multi-scale features derived from persistent homology, have seen increasing use in the machine learning community (Carrière et al., 2019, Guss & Salakhutdinov, 2018, Hofer et al., 2017, 2019a,b, Ramamurthy et al., 2019, Reininghaus et al., 2015, Rieck et al., 2019a,b), employing topology *directly* as a constraint for modern deep learning methods remains a challenge. This is due to the inherently discrete nature of these computations, making backpropagation through the computation of topological signatures immensely difficult or only possible in certain special circumstances (Chen et al., 2019, Hofer et al., 2019a, Poulenard et al., 2018).

This work presents a novel approach that permits obtaining gradients during the computation of topological signatures. This makes it possible to employ topological constraints while training deep neural networks, as well as building topology-preserving autoencoders. Specifically, we make the following contributions:

1. We develop a new topological loss term for autoencoders that helps harmonise the topology of the data space with the topology of the latent space.
2. We prove that our approach is stable on the level of mini-batches, resulting in suitable approximations of the persistent homology of a data set.
3. We empirically demonstrate that our loss term aids in dimensionality reduction by preserving topological structures in data sets; in particular, the learned latent representations are useful in that the preservation of topological structures can improve interpretability.

2. Background: Persistent Homology

Persistent homology (Barannikov, 1994, Edelsbrunner & Harer, 2008) is a method from the field of computational topology, which develops tools for analysing topological features (connectivity-based features such as connected components) of data sets. We first introduce the underlying concept of simplicial homology. For a simplicial complex $\mathfrak{K}$, i.e. a generalised graph with higher-order connectivity information such as cliques, simplicial homology employs matrix reduction algorithms to assign $\mathfrak{K}$ a family of groups, the *homology groups*. The d^{th} homology group $\mathrm{H}_d(\mathfrak{K})$ of $\mathfrak{K}$ contains d-dimensional topological features, such as connected components ($d = 0$), cycles/tunnels ($d = 1$), and voids ($d = 2$). Homology groups are typically summarised by their ranks, thereby obtaining a simple invariant "signature" of a manifold. For example, a circle in $\mathbb{R}^2$ has one feature with $d = 1$ (a cycle), and one feature with $d = 0$ (a connected component).

In practice, the underlying manifold $\mathbb{M}$ is unknown and we are working with a point cloud $X := \{x_1, \ldots, x_n\} \subseteq \mathbb{R}^d$ and a metric $\mathrm{dist}\colon X \times X \to \mathbb{R}$ such as the Euclidean distance. Persistent homology extends simplicial homology to this setting: instead of approximating $\mathbb{M}$ by means of a *single* simplicial complex, which would be an unstable procedure due to the discrete nature of X, persistent homology tracks changes in the homology groups over *multiple* scales of the metric. This is achieved by constructing a special simplicial complex, the Vietoris–Rips complex (Vietoris, 1927). For $0 \leq \epsilon < \infty$, the Vietoris–Rips complex of X at scale ϵ, denoted by $\mathfrak{R}_\epsilon(X)$, contains all

[†]Equal contribution. [‡]These authors jointly directed this work. [1]Department of Biosystems Science and Engineering, ETH Zurich, 4058 Basel, Switzerland [2]SIB Swiss Institute of Bioinformatics, Switzerland. Correspondence to: Karsten Borgwardt <karsten.borgwardt@bsse.ethz.ch>.

Proceedings of the 37th International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020. Copyright 2020 by the author(s).

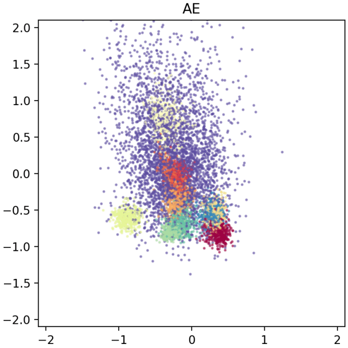
Michael Moor
Michael_D_Moor

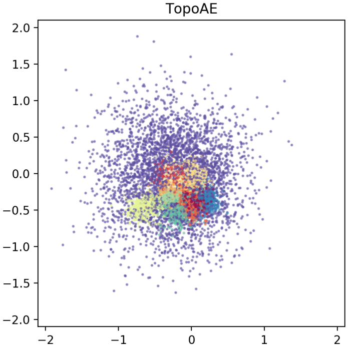
Max Horn
ExpectationMax

Karsten Borgwardt
kmborgwardt

Topological autoencoders

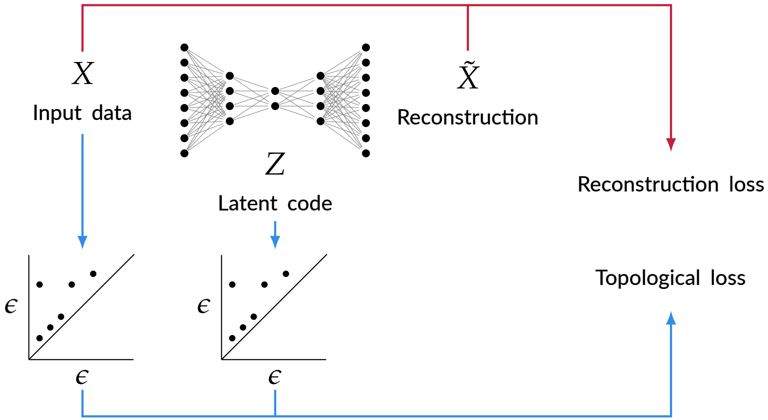
Motivation





Topological autoencoders

Overview



Topological autoencoders

Main intuition

Align persistence diagrams of an *input batch* and of a *latent batch* using a loss function!

Why this works in theory

Let X be a point cloud of cardinality n and $X^{(m)}$ be one subsample of X of cardinality m, i.e. $X^{(m)} \subseteq X$, sampled without replacement. We can bound the probability of the persistence diagrams of $X^{(m)}$ exceeding a threshold in terms of the bottleneck distance as

$$\mathbb{P}\left(\mathrm{W}_\infty\left(\mathcal{D}^X, \mathcal{D}^{X^{(m)}}\right) > \epsilon\right) \leq \mathbb{P}\left(\mathrm{dist}_\mathrm{H}\left(X, X^{(m)}\right) > 2\epsilon\right),$$

where dist_H denotes the Hausdorff distance. In other words: *mini-batches are topologically similar if the subsampling is not too coarse*.

Topological autoencoders

Gradient calculation intuition

Distance matrix $\mathbf{A}$

$$\begin{bmatrix} 0 & 1 & 9 & 10 \\ 1 & 0 & 7 & 8 \\ 9 & 7 & 0 & 3 \\ 10 & 8 & 3 & 0 \end{bmatrix}$$

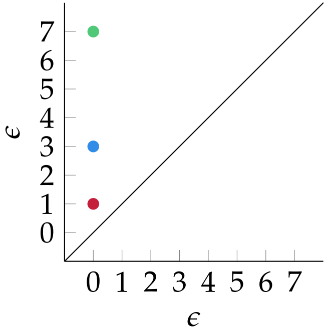
Every point in the persistence diagram can be mapped to *one* entry in the distance matrix! Each entry *is* a distance, so it can be changed during training (at least in the latent space).

Topological autoencoders

Gradient calculation intuition

Distance matrix $\mathbf{A}$

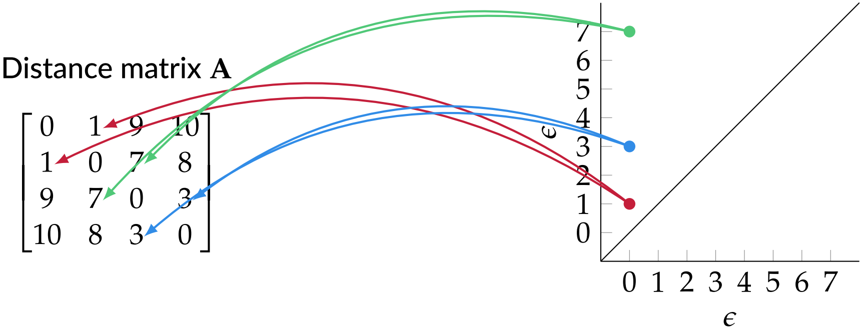
$$\begin{bmatrix} 0 & 1 & 9 & 10 \\ 1 & 0 & 7 & 8 \\ 9 & 7 & 0 & 3 \\ 10 & 8 & 3 & 0 \end{bmatrix}$$



Every point in the persistence diagram can be mapped to *one* entry in the distance matrix! Each entry *is* a distance, so it can be changed during training (at least in the latent space).

Topological autoencoders

Gradient calculation intuition



Every point in the persistence diagram can be mapped to *one* entry in the distance matrix! Each entry *is* a distance, so it can be changed during training (at least in the latent space).

Topological autoencoders

Loss term

$$\mathcal{L}_t := \mathcal{L}_{\mathcal{X}\to\mathcal{Z}} + \mathcal{L}_{\mathcal{Z}\to\mathcal{X}}$$

$$\mathcal{L}_{\mathcal{X}\to\mathcal{Z}} := \frac{1}{2}\left\|\mathbf{A}^X\left[\pi^X\right] - \mathbf{A}^Z\left[\pi^X\right]\right\|^2 \qquad \mathcal{L}_{\mathcal{Z}\to\mathcal{X}} := \frac{1}{2}\left\|\mathbf{A}^Z\left[\pi^Z\right] - \mathbf{A}^X\left[\pi^Z\right]\right\|^2$$

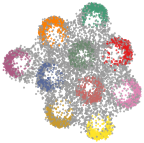
- $\mathcal{X}$: input space
- $\mathcal{Z}$: latent space
- $\mathbf{A}^X$: distances in input mini-batch
- $\mathbf{A}^Z$: distances in latent mini-batch
- π^X: persistence pairing of input mini-batch
- π^Z: persistence pairing of latent mini-batch

The loss is *bi-directional*!

Qualitative evaluation

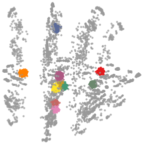
‘Spheres’ data set

PCA

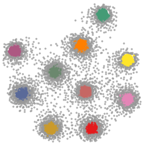


UMAP

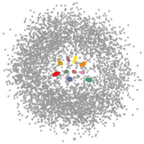
Autoencoder



Isomap



t-SNE



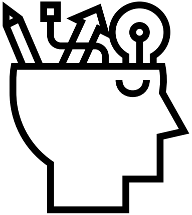
Topological autoencoder

Quantitative evaluation

Data set	Method	$\mathrm{KL}_{0.01}$	$\mathrm{KL}_{0.1}$	KL_{1}	ℓ-MRRE	ℓ-Cont	ℓ-Trust	ℓ-RMSE	MSE (data)
'Spheres'	Isomap	0.181	**0.420**	**0.00881**	**0.246**	**0.790**	**0.676**	10.4	
	PCA	0.332	0.651	0.01530	0.294	0.747	0.626	11.8	0.9610
	t-SNE	**0.152**	0.527	0.01271	**0.217**	0.773	**0.679**	**8.1**	
	UMAP	0.157	0.613	0.01658	0.250	0.752	0.635	**9.3**	
	AE	0.566	0.746	0.01664	0.349	0.607	0.588	13.3	**0.8155**
	TopoAE	**0.085**	**0.326**	**0.00694**	0.272	**0.822**	0.658	13.5	**0.8681**
'Fashion-MNIST'	PCA	**0.356**	**0.052**	**0.00069**	0.057	0.968	0.917	**9.1**	0.1844
	t-SNE	0.405	0.071	0.00198	**0.020**	0.967	**0.974**	41.3	
	UMAP	0.424	0.065	0.00163	0.029	**0.981**	0.959	**13.7**	
	AE	0.478	0.068	0.00125	**0.026**	0.968	**0.974**	20.7	**0.1020**
	TopoAE	**0.392**	**0.054**	**0.00100**	0.032	**0.980**	0.956	20.5	**0.1207**
'MNIST'	PCA	0.389	0.163	0.00160	0.166	0.901	0.745	**13.2**	0.2227
	t-SNE	**0.277**	**0.133**	0.00214	**0.040**	0.921	**0.946**	22.9	
	UMAP	**0.321**	0.146	0.00234	**0.051**	**0.940**	**0.938**	**14.6**	
	AE	0.620	0.155	**0.00156**	0.058	0.913	0.937	18.2	**0.1373**
	TopoAE	0.341	**0.110**	**0.00114**	0.056	**0.932**	0.928	19.6	**0.1388**

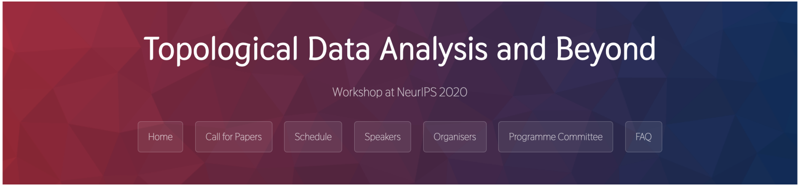
Open questions

A collection



- Should we *learn* filtrations or use *fixed* ones?
- Can we map topological features *back* to features in the data?
- How can we *scale* algorithms to massive data sets?

What is next?



- Visit the NeurIPS 2020 Workshop on 'Topological Data Analysis and Beyond'[3].
- Try out your own projects using `Giotto-tda`[4]
- Join the 'TDA in ML' Slack community!

[3] https://tda-in-ml.github.io
[4] https://giotto-ai.github.io/gtda-docs/latest/index.html

Take-away messages

- Topological features are incredibly versatile.
- Their integration in modern machine learning architectures is an ongoing research topic.
- Topological machine learning shines when working with *structural information*, such as in the case of graphs.

https://topology.rocks/ecml_pkdd_2020