

# Recent Advances in Topology-Based Graph Classification

Bastian Rieck

🐦 Pseudomanifold



**DBSSE**

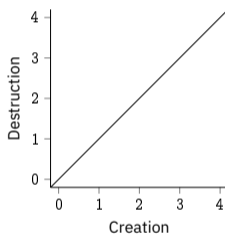
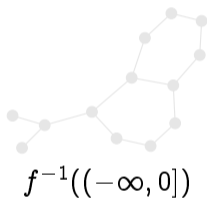
**ETH** zürich

# Setting the stage

*“Nerzhin, his lips tightly drawn, was inattentive to the point of rudeness; he did not even bother to ask what exactly Verenyov had written about this **arid branch of mathematics** in which he himself had done a little work for one of his courses. [...] Topology belonged to the stratosphere of human thought. It might conceivably turn out to be of some **use in the twenty-fourth century**, but for the time being...”*

# The persistent homology of unlabelled & unweighted graphs

Use a filtration based on *vertex degrees*. Set  $f(v) := \deg(v)$  for every vertex  $v$  and  $f(u, v) := \max(\deg(u), \deg(v))$  for every edge  $(u, v)$  to obtain  $f: \mathcal{G} \rightarrow \mathbb{R}$ .

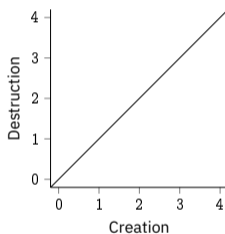
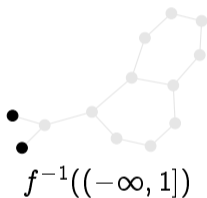


## Classification

Vectorise persistence diagrams using persistence images, for instance, and add them to your favourite machine learning pipeline.

# The persistent homology of unlabelled & unweighted graphs

Use a filtration based on *vertex degrees*. Set  $f(v) := \deg(v)$  for every vertex  $v$  and  $f(u, v) := \max(\deg(u), \deg(v))$  for every edge  $(u, v)$  to obtain  $f: \mathcal{G} \rightarrow \mathbb{R}$ .

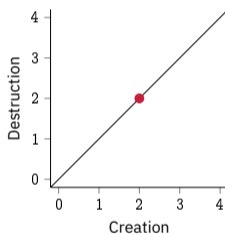
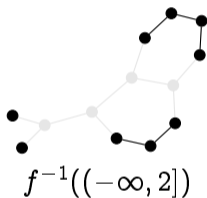


## Classification

Vectorise persistence diagrams using persistence images, for instance, and add them to your favourite machine learning pipeline.

# The persistent homology of unlabelled & unweighted graphs

Use a filtration based on *vertex degrees*. Set  $f(v) := \deg(v)$  for every vertex  $v$  and  $f(u, v) := \max(\deg(u), \deg(v))$  for every edge  $(u, v)$  to obtain  $f: \mathcal{G} \rightarrow \mathbb{R}$ .

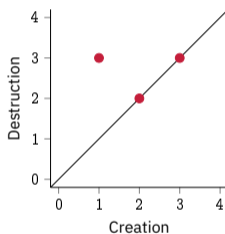
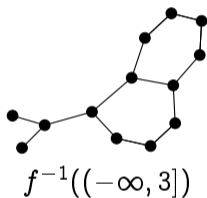


## Classification

Vectorise persistence diagrams using persistence images, for instance, and add them to your favourite machine learning pipeline.

# The persistent homology of unlabelled & unweighted graphs

Use a filtration based on *vertex degrees*. Set  $f(v) := \deg(v)$  for every vertex  $v$  and  $f(u, v) := \max(\deg(u), \deg(v))$  for every edge  $(u, v)$  to obtain  $f: \mathcal{G} \rightarrow \mathbb{R}$ .

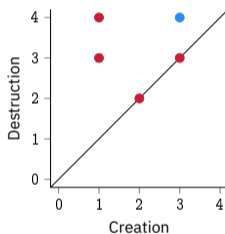
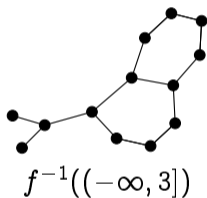


## Classification

Vectorise persistence diagrams using persistence images, for instance, and add them to your favourite machine learning pipeline.

# The persistent homology of unlabelled & unweighted graphs

Use a filtration based on *vertex degrees*. Set  $f(v) := \deg(v)$  for every vertex  $v$  and  $f(u, v) := \max(\deg(u), \deg(v))$  for every edge  $(u, v)$  to obtain  $f: \mathcal{G} \rightarrow \mathbb{R}$ .



## Classification

Vectorise persistence diagrams using persistence images, for instance, and add them to your favourite machine learning pipeline.

# Interlude

## The Weisfeiler–Lehman test for graph isomorphism

- 1 *Create* a colour for each node in the graph (based on its label or its degree).
- 2 *Collect* colours of adjacent nodes in a multiset.
- 3 *Compress* the colours in the multiset and the node's colour to form a new one.
- 4 *Continue* this relabelling scheme until the colours are stable.

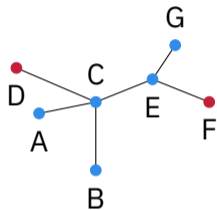
If the compressed labels of two graphs *diverge*, the graphs are *not* isomorphic!



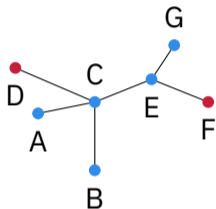
The other direction is not valid! Non-isomorphic graphs can give rise to coinciding compressed labels.



# Weisfeiler–Lehman iteration & feature vector

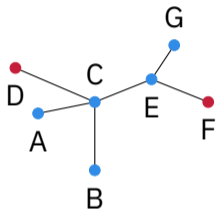


# Weisfeiler–Lehman iteration & feature vector



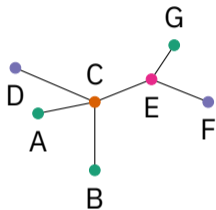
Node	Own label	Adjacent labels
A	●	●
B	●	●
C	●	● ● ● ●
D	●	●
E	●	● ● ●
F	●	●
G	●	●

# Weisfeiler–Lehman iteration & feature vector



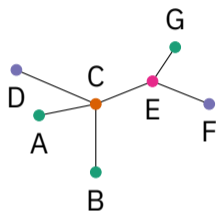
Node	Own label	Adjacent labels	Hashed label
A	●	●	●
B	●	●	●
C	●	● ● ● ●	●
D	●	●	●
E	●	● ● ●	●
F	●	●	●
G	●	●	●

# Weisfeiler–Lehman iteration & feature vector



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

# Weisfeiler–Lehman iteration & feature vector



Label	●	●	●	●
Count	3	1	2	1

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

Compare  $\mathcal{G}$  and  $\mathcal{G}'$  using a *kernel function*.

## Kernels

$$k(\mathcal{G}, \mathcal{G}') := \langle \Phi(\mathcal{G}), \Phi(\mathcal{G}') \rangle$$

$$k(\mathcal{G}, \mathcal{G}') := \exp\left(-\|\Phi(\mathcal{G}) - \Phi(\mathcal{G}')\|^2 / (2\sigma^2)\right)$$

# Making Weisfeiler–Lehman features persistent



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

**B. Rieck\***, C. Bock\* and K. Borgwardt, ‘A Persistent Weisfeiler–Lehman Procedure for Graph Classification’, *ICML*, 2019, pp. 5448–5458

## A Persistent Weisfeiler–Lehman Procedure for Graph Classification

Bastian Rieck\*, Christian Bock\*, Karsten Borgwardt\*

### Abstract

The Weisfeiler–Lehman graph kernel exhibits competitive performance on many graph classification tasks. However, in certain domains we are not able to capture structural components and cycles, topological features known for discriminating graphs. To extend such features, we have proposed topological label information and new, more expressive graph set metrics. This paper presents our proposed label features and topological information obtained using persistent homology, a concept from topological data analysis. Our methods exhibit the advantages of Weisfeiler–Lehman subgraph features, stable topological classification accuracy and its improvement in predictive performance as readily driven by including cycle information.

### 1. Introduction

Graph structural data sets are ubiquitous in a variety of different application domains, such as those posing a complex challenge while after requiring different tools to be solved. A common task involves graph classification, the which a variety of methods exist. These methods compare conventional neural networks (Ducrouot et al., 2015), or custom neural networks (Lu et al., 2017), or Hilbert space methods (Veličković et al., 2018), that have also been extended to graph levels. While several approaches for solving graph classification, the most common success the  $W$ -isomorphism framework (Weisfeiler, 1968), which enables it possible to define the similarity between two graphs as a function of the similarity of their substructures.

Algorithms that have been used for graph classification are easy from graphlets (Shervashidze et al., 2009), to small non-isomorphic graphs of fixed size, over directed hypergraphs (Department of Statistics, 2018) and graphlets (Dezobry, 2018), graphlet kernels (Borgwardt & Rieck, 2016), Weisfeiler–Lehman (Weisfeiler & Lehman, 1968) and Weisfeiler–Lehman (Weisfeiler & Lehman, 1968).

Proceedings of the 37th International Conference on Machine Learning, Long Beach, California, USA, 2019. Copyright 2019 by the authors.

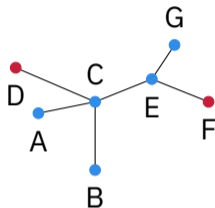
public (Borgwardt & Kröger, 2013), random walks (Liu et al., 2010; Rieck et al., 2011; Ingolese & Borgwardt, 2014). One of the most general approaches is the use of subgraph patterns (Raman & Chaturvedi, 2013), or patterns based on vertex subgraphs of a graph. These concepts naturally combine to make the topologically sensitive kernel used in Weisfeiler–Lehman (WL) graph kernel known as the Weisfeiler–Lehman (WL) graph kernel (Weisfeiler & Lehman, 1968; Shervashidze et al., 2011) we developed. Popular, recent, well-considered 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 discriminability of two graphs.

One of the disadvantages of this framework is that in a labelling step, to the step in which subgraph patterns are being compared to construct “node” labels are only compared with a fixed kernel, making them discriminability a static function. Moreover, the subgraph feature vector only contains counts of composed labels and can neither capture their relations with respect to the topology of the graph nor capture structural components and cycles, both of which are important and necessary features for discriminating graphs (Bock et al., 2018; Ingolese et al., 2017). We here propose an enhancement of the original WL substructure procedure that can assess subgraphs in topological data analysis (Borgwardt, 2017) to obtain these scores. Our contribution are as follows:

We extend the definition of topological features (connected components and cycles) by graphs and use them to define a novel set of WL subgraph features, which we show to be a generalised version of the original ones. We also propose a homology-based method to assess the similarity of the WL substructure procedure to classify non-isomorphic graphs. We demonstrate that our proposed feature performs well on many graph classification benchmark datasets. In particular, we empirically show that the topological homology-based method outperforms the original WL substructure procedure on many datasets, even on the set of all nodes.

# A distance between label multisets

Example for  $p = 1$



$$\begin{aligned}\text{dist}_{\mathfrak{M}}(C, E) &= \text{dist}_{\mathfrak{M}}(\{\bullet^3, \bullet^1\}, \{\bullet^2, \bullet^1\}) \\ &= \text{dist}_{\mathfrak{M}}([3, 1], [2, 1]) \\ &= 1\end{aligned}$$

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

$$\text{dist}_V(v_i, v_j) := \left[ l_{v_i}^{(h-1)} \neq l_{v_j}^{(h-1)} \right] + \text{dist}_{\mathfrak{M}}(l_{v_i}^{(h)}, l_{v_j}^{(h)}) + \tau$$

This turns *any* labelled graph into a weighted graph whose persistent homology we can calculate!

# Persistence-based Weisfeiler–Lehman algorithm

- 1 Perform Weisfeiler–Lehman iteration for a number of steps.
- 2 In each step, obtain a filtration using the vertex distance.
- 3 Store persistence-based features.

## Connected components

$$\Phi_{\text{P-WL}}^{(h)} := \left[ \mathfrak{p}^{(h)}(l_0), \mathfrak{p}^{(h)}(l_1), \dots \right]$$
$$\mathfrak{p}^{(h)}(l_i) := \sum_{l(v)=l_i} \text{pers}(v)^p,$$

## Cycles

$$\Phi_{\text{P-WL-C}}^{(h)} := \left[ \mathfrak{z}^{(h)}(l_0), \mathfrak{z}^{(h)}(l_1), \dots \right]$$
$$\mathfrak{z}^{(h)}(l_i) := \sum_{l_i \in l(u,v)} \text{pers}(u, v)^p,$$



# Classification results & summary

	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

# Learning graph filtrations

## Graph Filtration Learning

Christoph D. Hofer<sup>1</sup> Florian Graf<sup>2</sup> Bastian Rieck<sup>3</sup> Marc Niethammer<sup>3</sup> Roland Kwitt<sup>1</sup>

### Abstract

We propose an approach to learning with graph-structured data in the problem domain of graph classification. In particular, we present a novel type of readout operation to aggregate node features into a graph-level representation. To this end, we leverage persistent homology computed via a real-valued, learnable, filter function. We establish the theoretical foundation for differentiating through the persistent homology computation. Especially, we show that this type of readout operation compares favorably to previous techniques, especially when the graph connectivity structure is informative for the learning problem.



Figure 1. Overview of the proposed homological readout. Given a graphological complex, we use a real-valued graph functional  $f$  to assign a real-valued score to each node. A practical choice is to implement  $f$  as a CNN, with one level of message passing. We then compute persistent homology  $H_f$  using the filtration induced by  $f$ . Finally, features are fed through a readout function  $R$  and passed to a classifier (e.g., an MLP). Our approach allows passing a learning signal through the persistent homology computation, allowing to optimize  $f$  for the classification task.

### 1. Introduction

We consider the task of learning a function from the space of (finite) undirected graphs,  $\mathcal{G}$ , to a (discrete/continuous) target domain  $\mathcal{Y}$ . Additionally, graphs might have discrete, or continuous attributes attached to each node. Prominent examples for this class of learning problem appear in the context of classifying molecular structures, chemical compounds or social networks.

A substantial amount of research has been devoted to developing techniques for supervised learning with graph-structured data, ranging from kernel-based methods (Shervashidze et al., 2009, 2011; Feigen et al., 2013; Krage et al., 2016), to more recent approaches based on graph neural networks (GNN) (Scarselli et al., 2009; Hamilton et al., 2017; Zhang et al., 2018; Morris et al., 2019; Xu et al., 2019; You et al., 2020). Most of the latter works use an iterative message passing scheme (Gilmer et al., 2017) to learn node representations, followed by a graph-level pooling operation that aggregates node-level features. This

aggregation step is typically referred to as a readout operation. While research has mostly focused on variants of the message passing function, the readout step may have a significant impact, as it aims to capture properties of the entire graph. Importantly, both simple and more refined readout operations, such as summation, differentiable pooling (Viny et al., 2018), or sort pooling (Zhang et al., 2018), are inherently coupled to the amount of information carried over via multiple rounds of message passing. Hence, architectural GNN choices are typically guided by dataset characteristics, e.g., inquiring to tune the number of message passing rounds to the expected size of graphs.

**Contribution.** We propose a homological readout operation that captures the full global structure of a graph, while relying only on node representations learned from immediate neighbors. This not only alleviates the aforementioned design challenge, but potentially offers additional discriminative information. Similar to previous works, we consider a graph,  $\mathcal{G}$ , as a simplicial complex,  $K$ , and use persistent homology (Edelsbrunner & Harer, 2010) to capture homological changes that occur when constructing the graph one part at a time (i.e., revealing changes in the number of connected components or loops). As this hinges on an ordering of the pairs, prior works rely on a suitable filter function



Christoph Hofer



Florian Graf



Marc Niethammer

Twitter: [@MarcNiethammer](#)



Roland Kwitt

Twitter: [@rkwitt1982](#)

<sup>1</sup>Department of Computer Science, Univ. of Salzburg, Austria; <sup>2</sup>Department of Information Science and Engineering, ETH Zürich, Switzerland; <sup>3</sup>Univ. of North Carolina, Chapel Hill, USA. Correspondence to: Christoph D. Hofer (c.d.hofer@uni-salzburg.ac.at).

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

C. D. Hofer, F. Graf, **B. Rieck**, M. Niethammer and R. Kwitt, ‘Graph Filtration Learning’, *ICML*, 2020, arXiv: 1905.10996 [cs.LG]

# Graph neural networks in a nutshell

- ☆ Learn node representations  $h_v$  based on aggregated attributes  $a_v$
- ☆ Aggregate them over neighbourhoods
- ☆ Iteration  $k$  contains information up to  $k$  hops away
- ☆ Repeat iteration  $K$  times

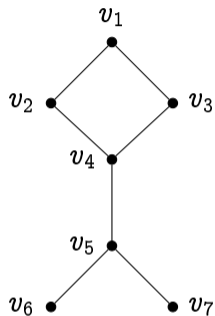
$$a_v^{(k)} := \text{aggregate}^{(k)} \left( \left\{ h_u^{(k-1)} \mid u \in \mathcal{N}_G(v) \right\} \right)$$

$$h_v^{(k)} := \text{combine}^{(k)} \left( h_v^{(k-1)}, a_v^{(k)} \right)$$

$$h_G := \text{readout} \left( \left\{ h_v^{(K)} \mid v \in V_G \right\} \right)$$

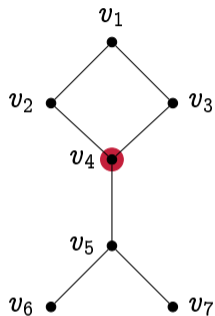
This terminology follows K. Xu, W. Hu, J. Leskovec and S. Jegelka, ‘How Powerful are Graph Neural Networks?’, *ICLR*, 2019.

# Message passing in graphs



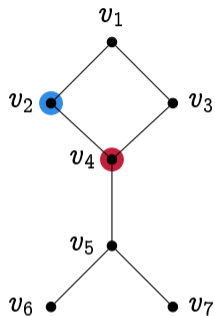
Here,  $v_i \in \mathbb{R}^d$  is a  $d$ -dimensional attribute vector (use one-hot encoding for labels).

# Message passing in graphs



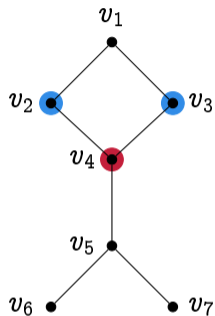
Here,  $v_i \in \mathbb{R}^d$  is a  $d$ -dimensional attribute vector (use one-hot encoding for labels).

# Message passing in graphs



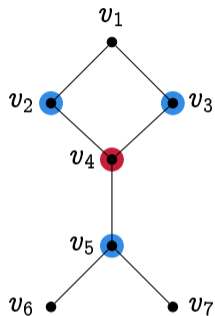
Here,  $v_i \in \mathbb{R}^d$  is a  $d$ -dimensional attribute vector (use one-hot encoding for labels).

# Message passing in graphs



Here,  $v_i \in \mathbb{R}^d$  is a  $d$ -dimensional attribute vector (use one-hot encoding for labels).

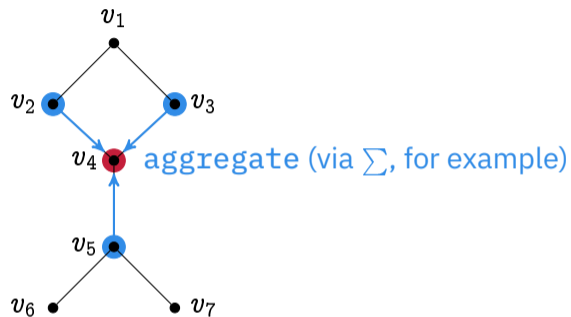
# Message passing in graphs



Here,  $v_i \in \mathbb{R}^d$  is a  $d$ -dimensional attribute vector (use one-hot encoding for labels).

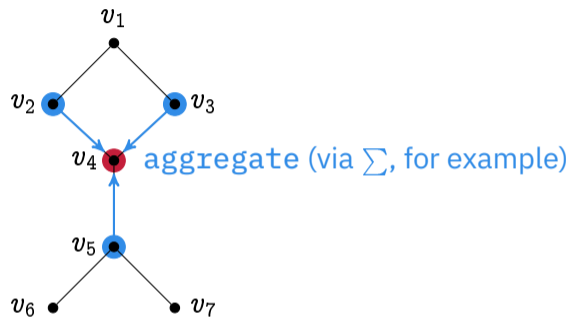


# Message passing in graphs



Here,  $v_i \in \mathbb{R}^d$  is a  $d$ -dimensional attribute vector (use one-hot encoding for labels).

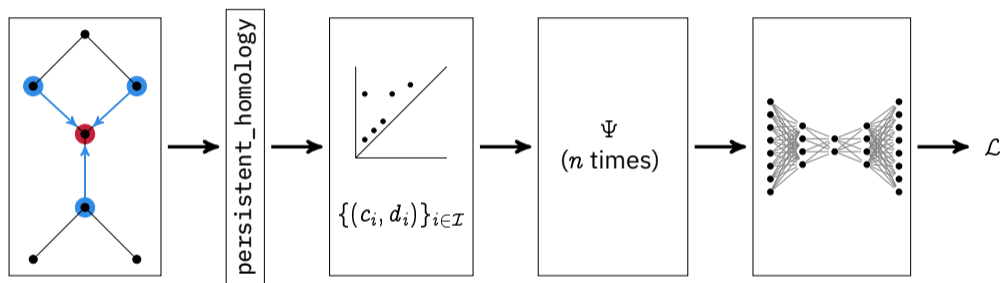
# Message passing in graphs



Here,  $v_i \in \mathbb{R}^d$  is a  $d$ -dimensional attribute vector (use one-hot encoding for labels).

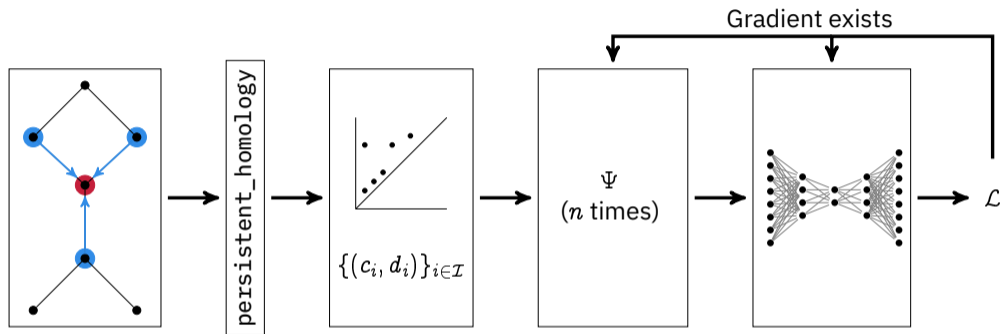
*Repeat* this process multiple times and update the vertex representations accordingly. Use a readout function to obtain a graph-level representation.

# A readout function based on persistent homology



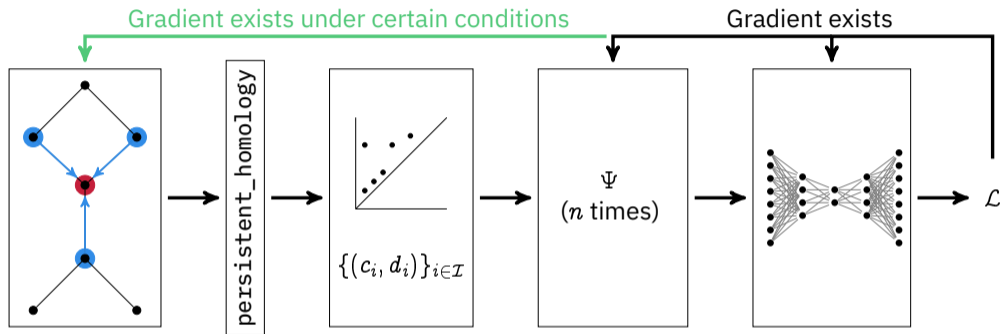
$\Psi: \mathcal{D} \rightarrow \mathbb{R}$  is a *differentiable* coordinatisation (embedding) function. By stacking  $n$  copies of  $\Psi$ , with different embedding parameters, we obtain an embedding into  $\mathbb{R}^n$ .

# A readout function based on persistent homology



$\Psi: \mathcal{D} \rightarrow \mathbb{R}$  is a *differentiable* coordinatisation (embedding) function. By stacking  $n$  copies of  $\Psi$ , with different embedding parameters, we obtain an embedding into  $\mathbb{R}^n$ .

# A readout function based on persistent homology



$\Psi: \mathcal{D} \rightarrow \mathbb{R}$  is a *differentiable* coordinatisation (embedding) function. By stacking  $n$  copies of  $\Psi$ , with different embedding parameters, we obtain an embedding into  $\mathbb{R}^n$ .

# Graph filtration learning in practice

- ☆ If  $f$  is *injective* on the graph vertices, the gradient exists.
- ☆ We can initialise  $f$  using the vertex degree or uniform weights (plus a symbolic perturbation to ensure gradient existence).
- ☆ Simple integration into existing architectures.

Method	IMDB-BINARY	IMDB-MULTI
<b>1-GIN (GFL)</b>	<b>74.5 ± 4.6</b>	49.7 ± 2.9
1-GIN (SUM)	73.5 ± 3.8	50.3 ± 2.6
1-GIN (SP)	73.0 ± 4.0	50.5 ± 2.1
Only node features	72.7 ± 4.6	49.9 ± 4.0
PH	68.9 ± 3.5	46.1 ± 4.2

# Topological layers for graph classification

arXiv:2102.07835v1 [cs.LG] 15 Feb 2021

## Topological Graph Neural Networks

Max Horn<sup>1,2</sup>, Edward De Brouwer<sup>1,2</sup>, Michael Moor<sup>1,2</sup>, Yves Moreau<sup>1</sup>, Bastian Rieck<sup>1,2,\*</sup>, and Karsten Borgwardt<sup>1,2</sup>

<sup>1</sup>Department of Information Systems and Engineering, ETH Zurich, 8092 Basel, Switzerland  
<sup>2</sup>IDS Bern Institute of Data Science, Switzerland  
RIECK@IDS.BN.CH, LUYKEN@IDS.BN.CH  
\*These authors contributed equally  
†These authors jointly supervised this work.

### Abstract

Graph neural networks (GNNs) are a powerful architecture for tackling graph learning tasks, yet have been shown to be inefficient to train on certain substructures, such as cycles. We present TOGL, a novel layer that incorporates global topology information of a graph using persistent homology. TOGL can be easily integrated into any type of GNN and is able to map representations to a space of the Weisfeiler-Leman test of linear phase. Empirically, GNNs with our layer learn to handle complex substructures with better performance than their counterparts. We show that TOGL can be used to classify graphs by homomorphisms but not by ordinary GNNs and on non-world data.

### 1. Introduction

Graphs are a natural description of structured data sets in many domains, including bioinformatics, image processing, and network analysis. Numerous methods address graph learning problems such as graph classification or node classification. Graph neural networks (GNNs) describe a flexible set of architectures for graph learning tasks and have seen many successful applications over recent years. In their core, most GNNs are based on iterative message passing schemes. Since these schemes are relying fundamentally on neighborhood information, GNNs cannot necessarily capture certain complex topological structures in graphs, such as cycles.

By contrast, methods based on topological features, commonly summarized under the umbrella term of topological data analysis (TDA), have shown promising results in our share learning tasks. Focusing on coarse structures—such as the presence or absence of cycles—they can be used to provide multi-scale representations that capture the shape of complex structural and nonstructural data sets. In this paper, we propose a [Topological Graph Layer \(TOGL\)](#) that can be easily integrated into any GNN or TDA pipeline. We then obtain a generic way to augment existing GNNs and increase their expressivity to graph learning tasks.

**Our contributions.** We describe a layer based on TDA and argue that our layer is integrated into any GNN. Our layer is differentiable and capable of learning differentiable topological representations of a graph. We prove that even by itself, our layer is strictly more expressive than any GNN since it incorporates the ability to work with multi-scale topological information in a graph. Moreover, we show that TOGL can improve predictive performance of a GNN when topological information is relevant for the task.

### 2. Background: Computational Topology

This paper considers undirected graphs, i.e., of the form  $G = (V, E)$  with a set of vertices  $V$  and a set of edges  $E \subseteq V \times V$ . A simple set of topological features of  $G$  is given by their number of connected components  $\beta_0$  and their number of cycles  $\beta_1$ . These counts are also known as the 0th and 1st Betti numbers or dimensions  $\beta_0$  and  $\beta_1$ , respectively, and can be computed efficiently (see, e.g., Edelsbrunner & Harer, 2010). Both counts are invariant under graph isomorphism; this is a consequence of a more general theorem in algebraic topology that incorporates higher-dimensional topological features as well (Battista, 2002, pp. 321–323). Thus, given two graphs  $G$  and  $G'$  with  $\beta_0 = \beta_0'$  and  $\beta_1 = \beta_1'$ , their Betti numbers will coincide. On such many statements that are computationally efficient to compute the presence relative dimension and level, leading their ability to distinguish between graphs. This is underlined by the fact that Betti numbers are stable under homeomorphism. It is possible to increase their expressivity by accounting the existence of a graph filtration, i.e., a sequence of nested subgraphs  $G_i$  of  $G$  such that

$$0 \subset G^0 \subset G^1 \subset G^2 \subset \dots \subset G^{n-1} \subset G^n = G. \quad (1)$$

A filtration makes it possible to obtain more insights into the graph by “monitoring” topological features of each step, including their topological relevance, which is referred to as their persistence. If a topological feature appears for the first time in  $G^i$  and disappears in  $G^j$ , we assign this feature a persistence of  $i - j$ . Equivalently, we can represent all of these features as a graph  $\beta$ , which is called its persistence diagram (D). In a similar sense diagrams, we represent a graph  $G$  as a set of such features as the ones that are essential for the respective Betti numbers. This process was formalized and extended to a wide class of structured data sets, namely complex complexes, and is known under the name of persistent



Max Horn  
ExpectationMax



Edward De Brouwer  
EdwardOnBrew



Michael Moor  
Michael\_D\_Moor



Yves Moreau

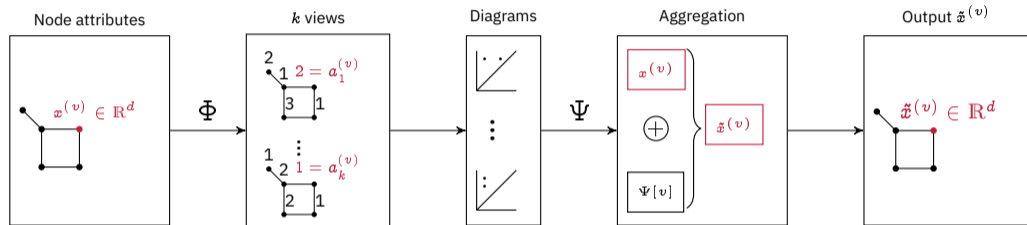


Karsten Borgwardt  
kmborgwardt

M. Horn\*, E. De Brouwer\*, M. Moor, Y. Moreau, **B. Rieck**\*<sup>†</sup> and K. Borgwardt<sup>†</sup>,  
*Topological Graph Neural Networks*, 2021, arXiv: 2102.07835 [cs.LG]

# Topological graph neural networks

## Overview

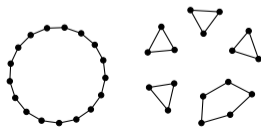
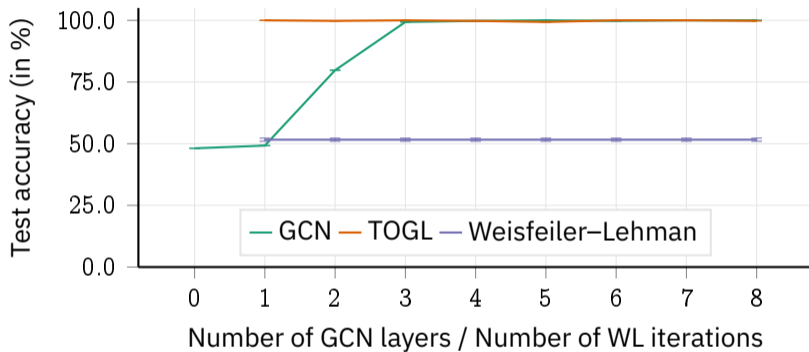


- ☆ Use a node map  $\Phi: \mathbb{R}^d \rightarrow \mathbb{R}^k$  to create  $k$  different filtrations of the graph.
- ☆ Use a coordinatisation function  $\Psi$  to create *compatible* representations of the node attributes.



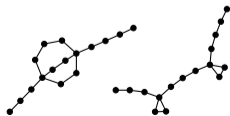
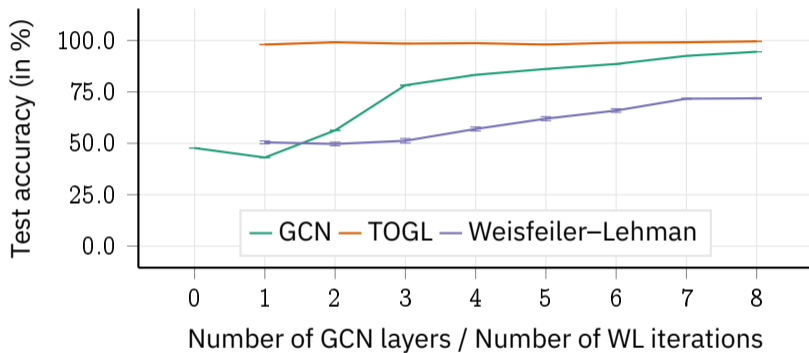
# Expressivity

Cycles data set



# Expressivity

## Necklaces data set



# Empirical results

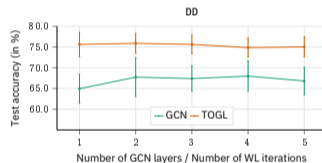
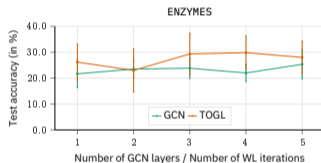
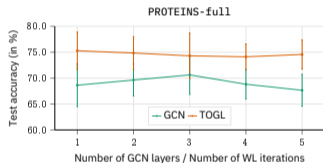
## Replacing one GCN layer with TOGL

Method	PROTEINS-full1	ENZYMES	DD	IMDB-BINARY	REDDIT-BINARY
GAT-4	$76.3 \pm 2.4$	$68.5 \pm 5.2$	$75.9 \pm 3.8$	—	—
GATED-GCN-4	$76.4 \pm 2.9$	$65.7 \pm 4.9$	$72.9 \pm 2.1$	—	—
GCN-4	$76.1 \pm 2.4$	$65.8 \pm 4.6$	$72.8 \pm 4.1$	$68.6 \pm 4.9$	$92.8 \pm 1.7$
GIN-4	$74.1 \pm 3.4$	$65.3 \pm 6.8$	$71.9 \pm 3.9$	$72.9 \pm 4.7$	$89.8 \pm 2.2$
<b>TopoGNN-3-1</b>	$76.0 \pm 3.9$	$53.0 \pm 9.2$	$73.2 \pm 4.7$	$72.0 \pm 2.3$	$89.4 \pm 2.2$
WL	$73.1 \pm 0.5$	$54.3 \pm 0.9$	$77.7 \pm 2.0$	$71.2 \pm 0.5$	$78.0 \pm 0.6$
WL-OA	$73.5 \pm 0.9$	$58.9 \pm 0.9$	$77.8 \pm 1.2$	$74.0 \pm 0.7$	$87.6 \pm 0.3$

The inclusion of global topological features has a slightly detrimental effect for some of the data sets! Why?

# Empirical results

What if we *drop* existing node features?



# Summary

- ☆ Topological features improve graph classification tasks.
- ☆ Recent advances result in *differentiable* representations.
- ☆ Often, the main performance drive is unclear; we need *ablation studies* that disentangle performance.
- ☆ *Hybrid* 🐙 models show particular promise for graph classification.

## ♥ Acknowledgements



MLCB @ ETH Zurich



TU Graz



KU Leuven