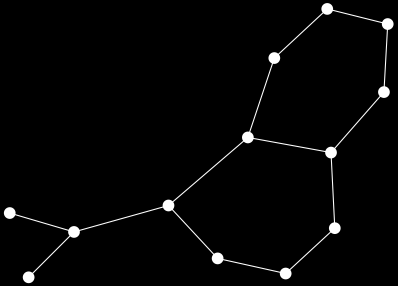
HELMHOLTZ MUNICH AIH Institute of AI for Health

Topology-Based Graph Learning

Graph Embeddings: Theory meets Practice

Bastian Rieck (@Pseudomanifold)

Graph learning



Tasks

- Graph classification
- Graph regression
- Node/edge classification
- Node/edge regression
- Link prediction

Graph representations

Fundamental properties

- Two graphs G and G' can have a *different* number of vertices.
- Hence, we require a *vectorised representation* $f\colon \mathbb{G} \to \mathbb{R}^d$ of graphs.
- Such a representation f needs to be *permutation-invariant*.

Now and then

Shallow approaches

- node2vec (encoder–decoder)
- Graph kernels (RKHS feature maps)
- Laplacian-based embeddings

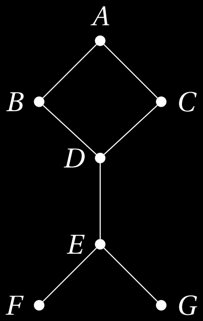
Deep approaches

- Graph convolutional networks
- Graph isomorphism networks
- Graph attention networks

Message passing

The predominant paradigm in graph machine learning

Neighbouring nodes can exchange *messages*. If this is *iterated*, messages can be ‘diffused’ to larger parts of the graph.

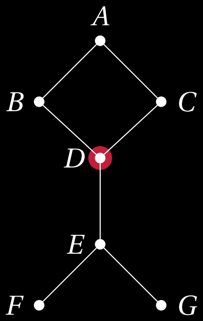


- ✩ Operations remain local.
- ✩ Message passing can be iterated.
- ✩ Need to define aggregation function.
- ✩ Representations can be combined.

Message passing

The predominant paradigm in graph machine learning

Neighbouring nodes can exchange *messages*. If this is *iterated*, messages can be 'diffused' to larger parts of the graph.

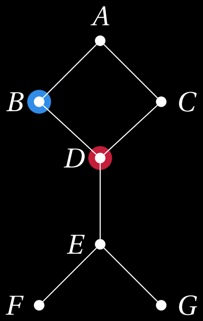


- Operations remain local.
- Message passing can be iterated.
- Need to define aggregation function.
- Representations can be combined.

Message passing

The predominant paradigm in graph machine learning

Neighbouring nodes can exchange *messages*. If this is *iterated*, messages can be 'diffused' to larger parts of the graph.

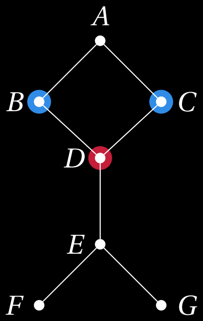


- Operations remain local.
- Message passing can be iterated.
- Need to define aggregation function.
- Representations can be combined.

Message passing

The predominant paradigm in graph machine learning

Neighbouring nodes can exchange *messages*. If this is *iterated*, messages can be ‘diffused’ to larger parts of the graph.

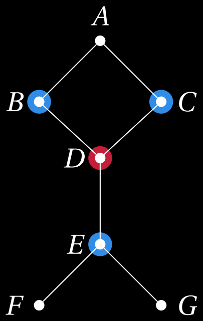


- Operations remain local.
- Message passing can be iterated.
- Need to define aggregation function.
- Representations can be combined.

Message passing

The predominant paradigm in graph machine learning

Neighbouring nodes can exchange *messages*. If this is *iterated*, messages can be 'diffused' to larger parts of the graph.

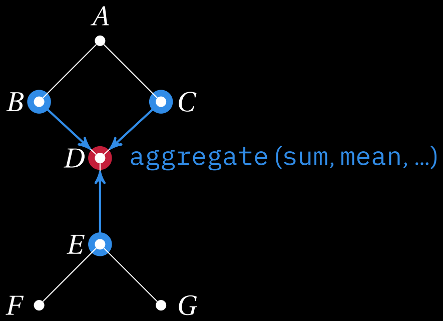


- Operations remain local.
- Message passing can be iterated.
- Need to define aggregation function.
- Representations can be combined.

Message passing

The predominant paradigm in graph machine learning

Neighbouring nodes can exchange *messages*. If this is *iterated*, messages can be 'diffused' to larger parts of the graph.



- Operations remain local.
- Message passing can be iterated.
- Need to define aggregation function.
- Representations can be combined.

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 procedure K times.

$$\begin{aligned}
a_v^{(k)} &:= \texttt{aggregate}^{(k)}\left(\left\{h_u^{(k-1)} \mid u \in \mathcal{N}_{\mathrm{G}}(v)\right\}\right)\\
h_v^{(k)} &:= \texttt{combine}^{(k)}\left(h_v^{(k-1)}, a_v^{(k)}\right)\\
h_{\mathrm{G}} &:= \texttt{readout}\left(\left\{h_v^{(K)} \mid v \in \mathcal{V}_{\mathrm{G}}\right\}\right)
\end{aligned}$$

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

Expressivity of graph neural networks

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!

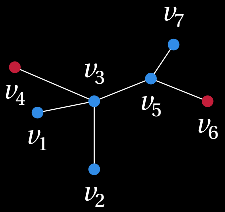
The other direction is not valid! Non-isomorphic graphs can give rise to coinciding compressed labels.

WL[1] is the baseline for measuring GNN expressivity.[1,2]

[1] C. Morris et al., 'Weisfeiler and Leman Go Neural: Higher-Order Graph Neural Networks', *AAAI*, 2019.
[2] K. Xu, W. Hu, J. Leskovec and S. Jegelka, 'How Powerful are Graph Neural Networks?', *ICLR*, 2019.

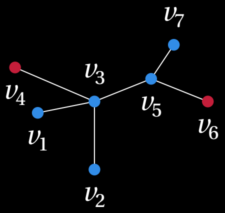
Weisfeiler–Lehman subtree features

Example for $h = 1$



Weisfeiler–Lehman subtree features

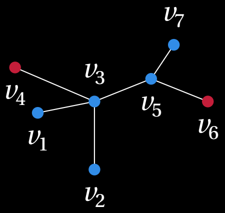
Example for $h = 1$



Node	Own label	Adjacent labels
v_1	🔵	🔵
v_2	🔵	🔵
v_3	🔵	🔵🔵🔵🔴
v_4	🔴	🔵
v_5	🔵	🔵🔵🔴
v_6	🔴	🔵
v_7	🔵	🔵

Weisfeiler–Lehman subtree features

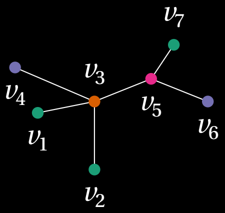
Example for $h = 1$



Node	Own label	Adjacent labels	Hashed label
v_1	🔵	🔵	🟢
v_2	🔵	🔵	🟢
v_3	🔵	🔵🔵🔵🔴	🟠
v_4	🔴	🔵	🟣
v_5	🔵	🔵🔵🔴	🩷
v_6	🔴	🔵	🟣
v_7	🔵	🔵	🟢

Weisfeiler–Lehman subtree features

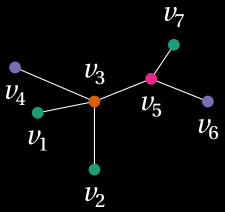
Example for $h = 1$



Node	Own label	Adjacent labels	Hashed label
v_1	🔵	🔵	🟢
v_2	🔵	🔵	🟢
v_3	🔵	🔵🔵🔵🔴	🟠
v_4	🔴	🔵	🟣
v_5	🔵	🔵🔵🔴	🩷
v_6	🔴	🔵	🟣
v_7	🔵	🔵	🟢

Weisfeiler–Lehman subtree features

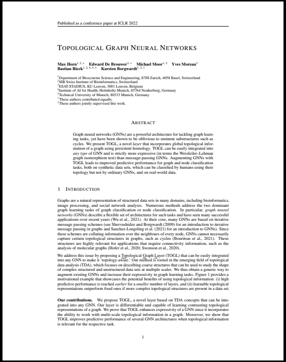
Example for $h = 1$



Label	●	●	●	●
Count	3	1	2	1
Feature vector	$\Phi(G) := (3, 1, 2, 1)$			

A topological layer for graph classification

M. Horn*, E. De Brouwer*, M. Moor, Y. Moreau, **B. Rieck**† and K. Borgwardt†, 'Topological Graph Neural Networks', *ICLR*, 2022



Topological Graph Neural Networks

Max Horn[1, 2, *] **Edward De Brouwer**[3, *] **Michael Moor**[1, 2] **Yves Moreau**[3]
Bastian Rieck[1, 2, 4, 5, †] **Karsten Borgwardt**[1, 2, †]

[1]Department of Biosystems Science and Engineering, ETH Zurich, 4058 Basel, Switzerland
[2]SIB Swiss Institute of Bioinformatics, Switzerland
[3]ESAT-STADIUS, KU Leuven, 3001 Leuven, Belgium
[4]Institute of AI for Health, Helmholtz Munich, 85764 Neuherberg, Germany
[5]Technical University of Munich, 80333 Munich, Germany
[*]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 oblivious to eminent substructures such as cycles. We present TOGL, a novel *layer* that incorporates global topological information of a graph using persistent homology. TOGL can be easily integrated into *any type* of GNN and is strictly more expressive (in terms the Weisfeiler–Lehman graph isomorphism test) than message-passing GNNs. Augmenting GNNs with TOGL leads to improved predictive performance for graph and node classification tasks, both on synthetic data sets, which can be classified by humans using their topology but not by ordinary GNNs, and on real-world data.

1 Introduction

Graphs are a natural representation of structured data sets in many domains, including bioinformatics, image processing, and social network analysis. Numerous methods address the two dominant graph learning tasks of graph classification or node classification. In particular, *graph neural networks* (GNNs) describe a flexible set of architectures for such tasks and have seen many successful applications over recent years (Wu et al., 2021). At their core, many GNNs are based on iterative message passing schemes (see Shervashidze and Borgwardt (2009) for an introduction to iterative message passing in graphs and Sanchez-Lengeling et al. (2021) for an introduction to GNNs). Since these schemes are collating information over the neighbours of every node, GNNs cannot necessarily capture certain topological structures in graphs, such as cycles (Bouritsas et al., 2021). These structures are highly relevant for applications that require connectivity information, such as the analysis of molecular graphs (Hofer et al., 2020; Swenson et al., 2020).

We address this issue by proposing a Topological Graph Layer (TOGL) that can be easily integrated into any GNN to make it 'topology-aware.' Our method is rooted in the emerging field of topological data analysis (TDA), which focuses on describing coarse structures that can be used to study the shape of complex structured and unstructured data sets at multiple scales. We thus obtain a generic way to augment existing GNNs and increase their expressivity in graph learning tasks. Figure 1 provides a motivational example that showcases the potential benefits of using topological information: (i) high predictive performance is reached *earlier* for a *smaller* number of layers, and (ii) learnable topological representations outperform fixed ones if more complex topological structures are present in a data set.

Our contributions. We propose TOGL, a novel layer based on TDA concepts that can be integrated into any GNN. Our layer is differentiable and capable of learning contrasting topological representations of a graph. We prove that TOGL enhances expressivity of a GNN since it incorporates the ability to work with multi-scale topological information in a graph. Moreover, we show that TOGL improves predictive performance of several GNN architectures when topological information is relevant for the respective task.

Max Horn
@ExpectationMax

Edward De Brouwer
@EdwardOnBrew

Michael Moor
@Michael_D_Moor

Yves Moreau

Karsten Borgwardt
@kmborgwardt

Motivation

Status quo

- Graphs are topological objects.
- But GNNs are *incapable* of recognising certain topological structures!

Challenge

What can we gain when imbuing them with knowledge about the topology?

Background

A brief introduction to persistent homology

Persistent homology is based on the concept of a *filtration*, i.e. an ordering of nodes. As nodes are added to the graph, its topological features change.

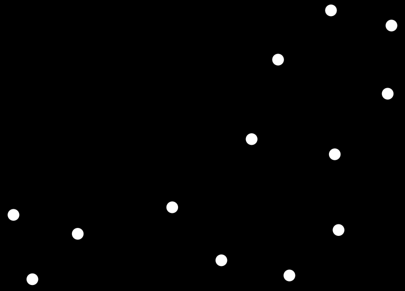
A hierarchy of topological features

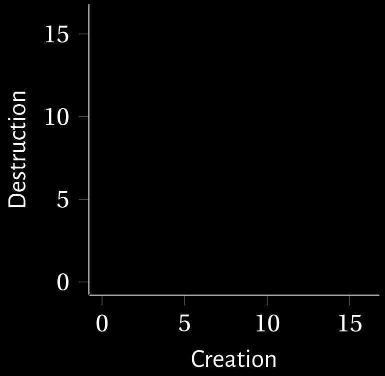
- $d = 0$: connected components
- $d = 1$: cycles
- $d = 2$: voids (requires representation of 2-cliques in graph)
- $d = D$: higher-dimensional holes (requires representation of D-cliques in graph)

Background

A brief introduction to persistent homology, continued

Store information about features in a *persistence diagram*. A tuple (c, d) indicates that a topological feature was created at step c and destroyed at step d.

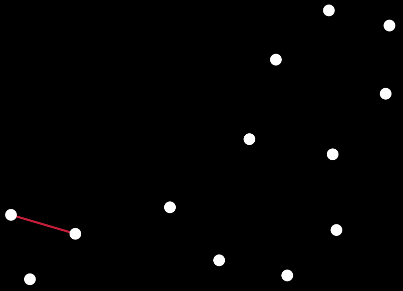


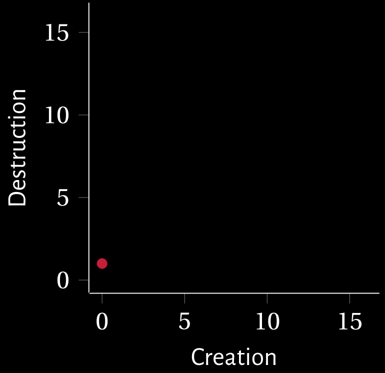


Background

A brief introduction to persistent homology, continued

Store information about features in a *persistence diagram*. A tuple (c, d) indicates that a topological feature was created at step c and destroyed at step d.

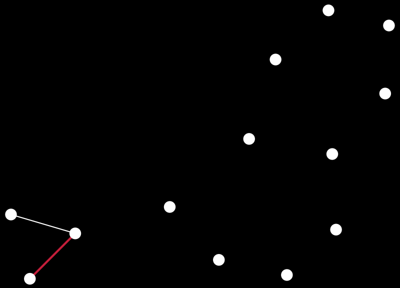


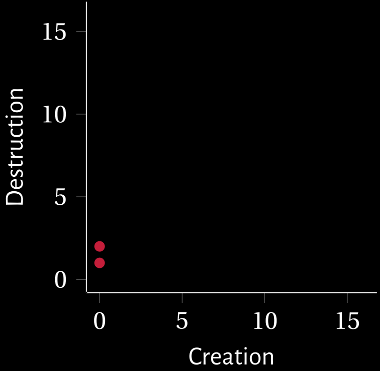


Background

A brief introduction to persistent homology, continued

Store information about features in a *persistence diagram*. A tuple (c, d) indicates that a topological feature was created at step c and destroyed at step d.

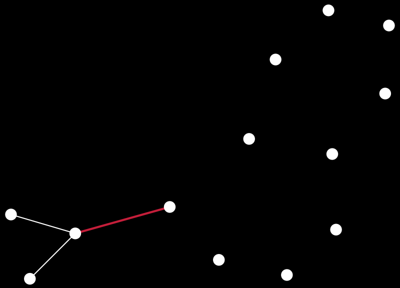


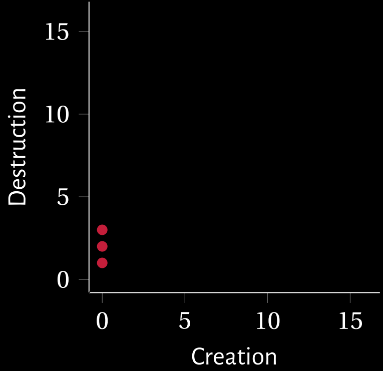


Background

A brief introduction to persistent homology, continued

Store information about features in a *persistence diagram*. A tuple (c, d) indicates that a topological feature was created at step c and destroyed at step d.

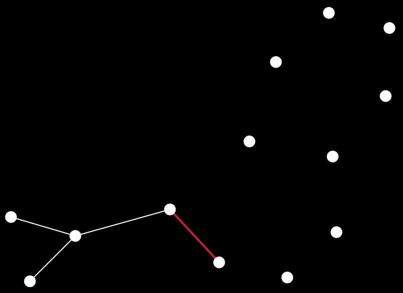


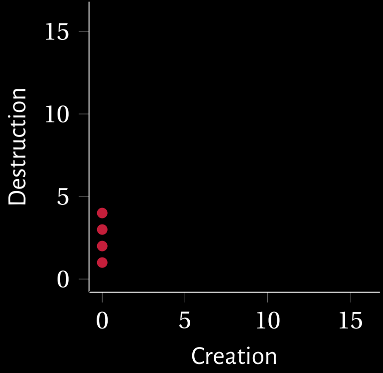


Background

A brief introduction to persistent homology, continued

Store information about features in a *persistence diagram*. A tuple (c, d) indicates that a topological feature was created at step c and destroyed at step d.

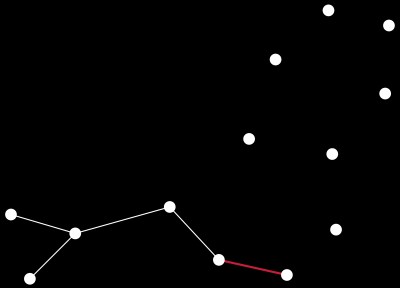


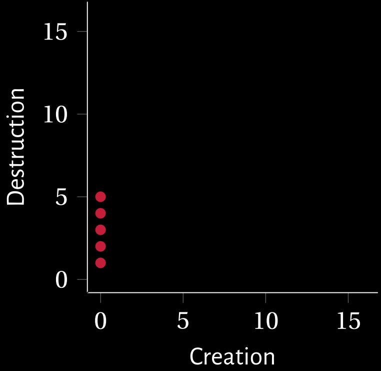


Background

A brief introduction to persistent homology, continued

Store information about features in a *persistence diagram*. A tuple (c, d) indicates that a topological feature was created at step c and destroyed at step d.

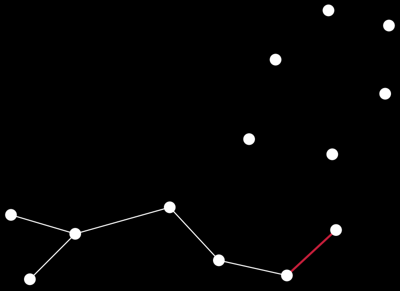


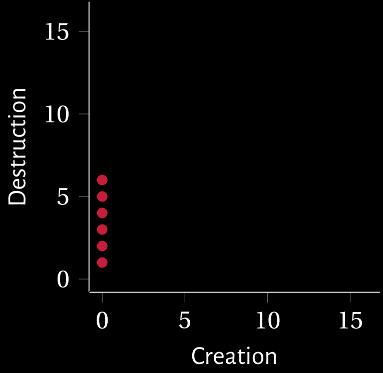


Background

A brief introduction to persistent homology, continued

Store information about features in a *persistence diagram*. A tuple (c, d) indicates that a topological feature was created at step c and destroyed at step d.

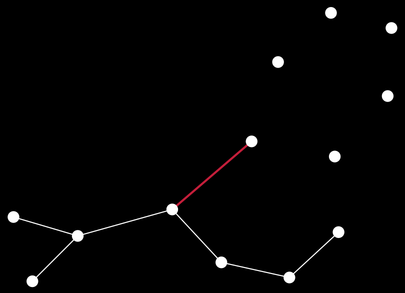


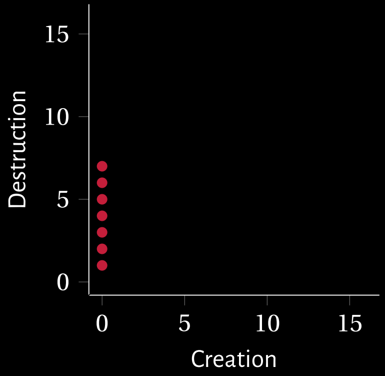


Background

A brief introduction to persistent homology, continued

Store information about features in a *persistence diagram*. A tuple (c, d) indicates that a topological feature was created at step c and destroyed at step d.

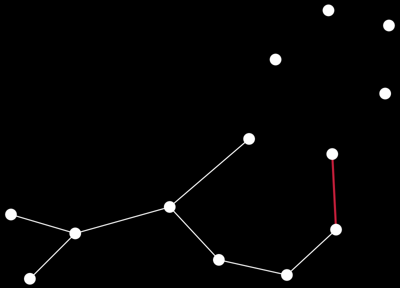


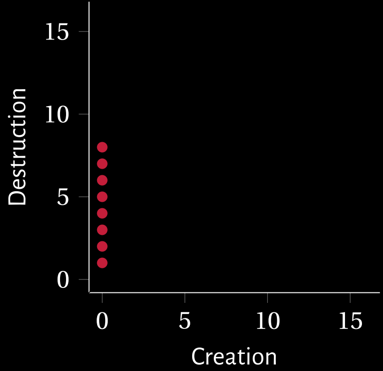


Background

A brief introduction to persistent homology, continued

Store information about features in a *persistence diagram*. A tuple (c, d) indicates that a topological feature was created at step c and destroyed at step d.

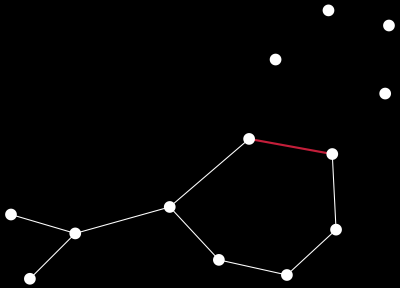


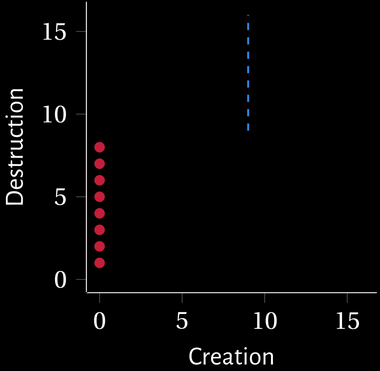


Background

A brief introduction to persistent homology, continued

Store information about features in a *persistence diagram*. A tuple (c, d) indicates that a topological feature was created at step c and destroyed at step d.

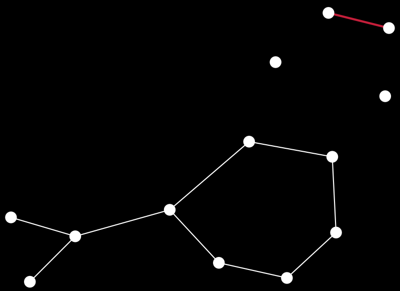


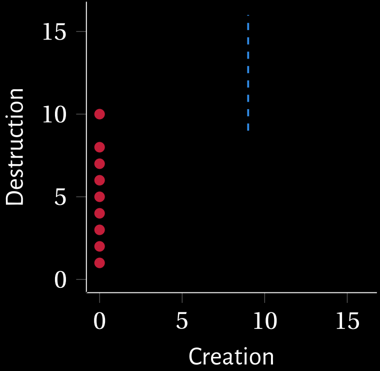


Background

A brief introduction to persistent homology, continued

Store information about features in a *persistence diagram*. A tuple (c, d) indicates that a topological feature was created at step c and destroyed at step d.

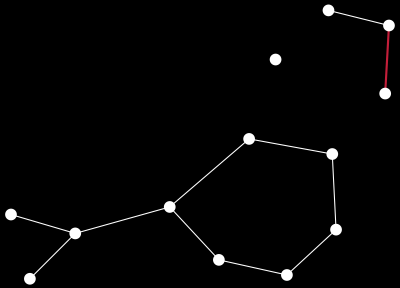


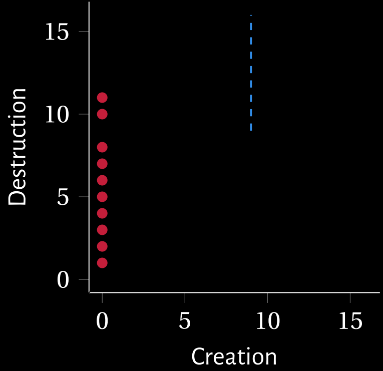


Background

A brief introduction to persistent homology, continued

Store information about features in a *persistence diagram*. A tuple (c, d) indicates that a topological feature was created at step c and destroyed at step d.

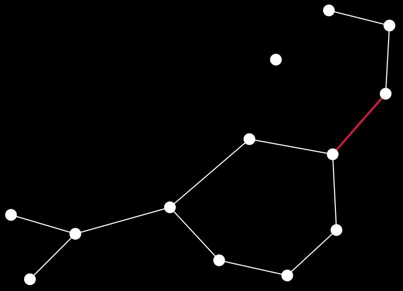


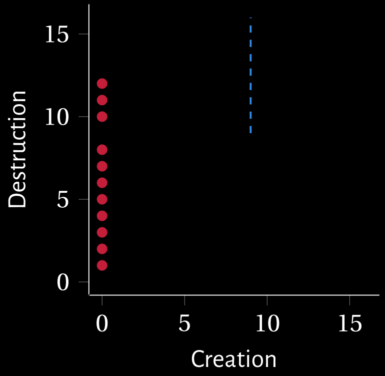


Background

A brief introduction to persistent homology, continued

Store information about features in a *persistence diagram*. A tuple (c, d) indicates that a topological feature was created at step c and destroyed at step d.

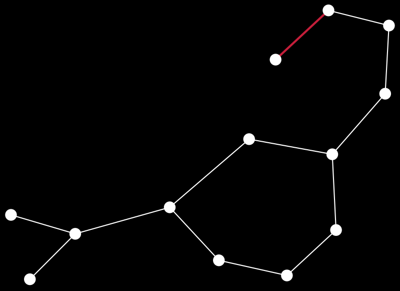


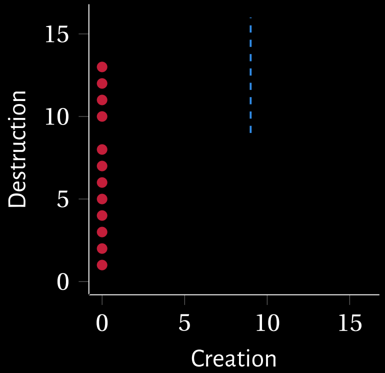


Background

A brief introduction to persistent homology, continued

Store information about features in a *persistence diagram*. A tuple (c, d) indicates that a topological feature was created at step c and destroyed at step d.

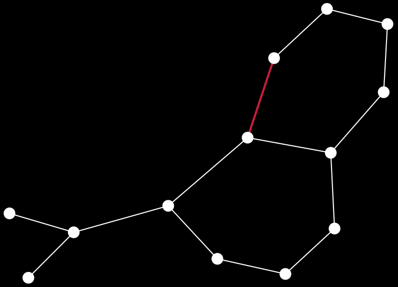


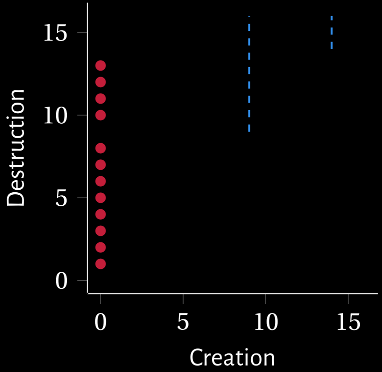


Background

A brief introduction to persistent homology, continued

Store information about features in a *persistence diagram*. A tuple (c, d) indicates that a topological feature was created at step c and destroyed at step d.

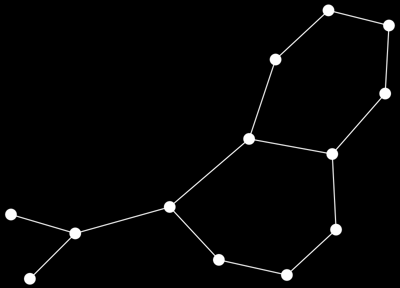


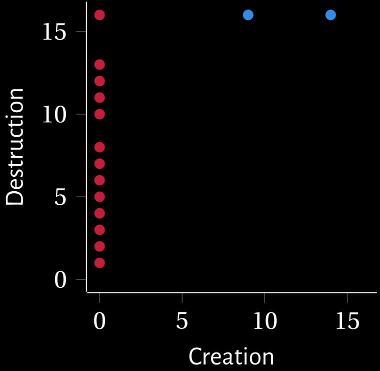


Background

A brief introduction to persistent homology, continued

Store information about features in a *persistence diagram*. A tuple (c, d) indicates that a topological feature was created at step c and destroyed at step d.





Taking stock

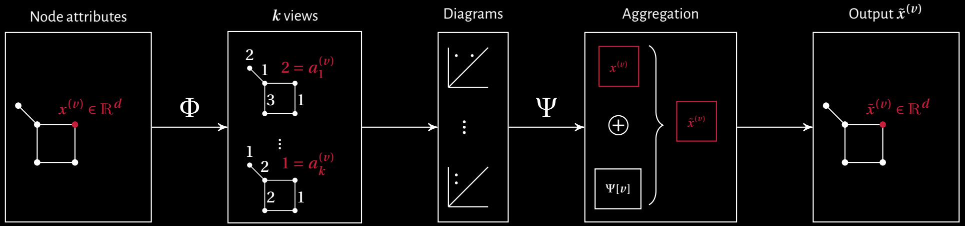
- Filtrations provide multi-scale topological features.
- Persistence diagrams serve as topological descriptors.

Questions

- How to obtain ‘good’ filtrations?
- How to use persistence diagrams (i.e. multi-sets) in a differentiable setting?

Topological graph neural networks

Overview



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

Choosing Φ and Ψ

- The node map Φ can be realised using a *neural network*.
- The coordinatisation function Ψ can be realised using *any* vectorisation of persistence diagrams (landscapes, images, …), but we found a *differentiable coordinatisation function* to be most effective.[3]

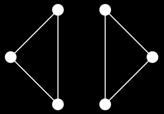
[3] C. D. Hofer, F. Graf, **B. Rieck**, M. Niethammer and R. Kwitt, 'Graph Filtration Learning', *ICML*, 2020.

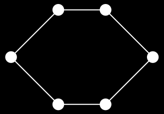
Expressivity of TOGL

Theorem

TOGL (and persistent homology) is ***more expressive*** *than WL[1], i.e. (i) if the WL[1] label sequences for two graphs G and G' diverge, there exists an injective filtration f such that the corresponding persistence diagrams $\mathscr{D}_0$ and $\mathscr{D}_0'$ are not equal, and (ii) there are graphs that WL[1] cannot distinguish but TOGL can!*

Example graphs


G


G′

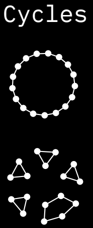
Experiments

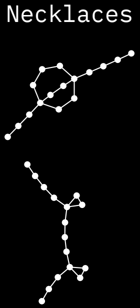
- Take existing GNN architecture.
- Replace one layer by TOGL.
- Measure predictive performance.

This strategy ensures that the number of parameters is approximately the same, thus facilitating a fair comparison!

Synthetic data sets

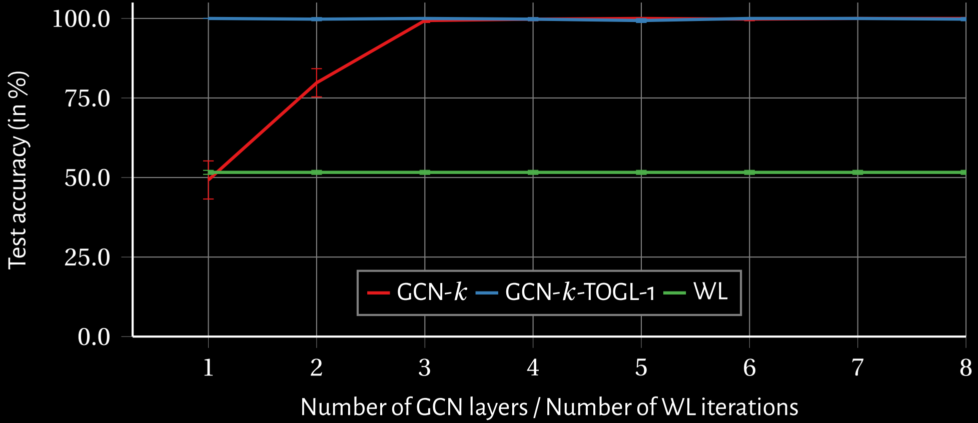
Binary classification problem; generate same number of graphs for each of the classes. Use simple topological structures that are nevertheless challenging to detect with standard GNNs.





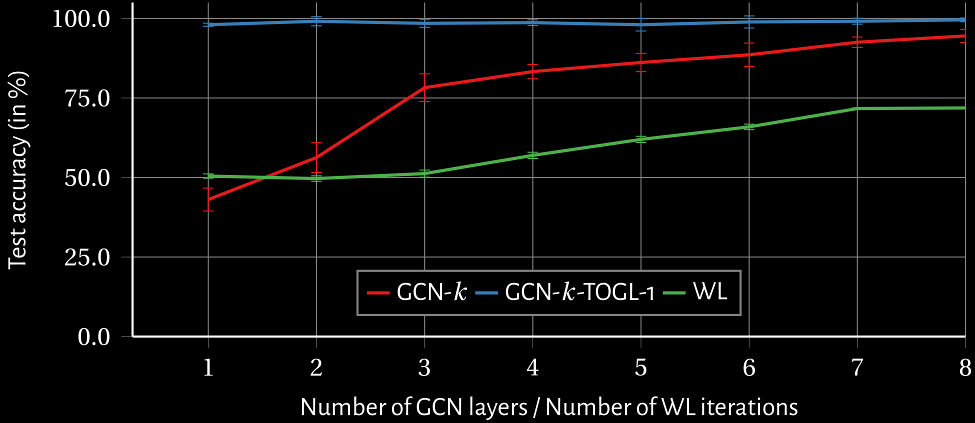
Expressivity

`Cycles` data set



Expressivity

`Necklaces` data set



Classifying graphs/nodes based on structural features alone

Existing data sets tend to 'leak' information into node attributes, thus decreasing the utility of topological features. Hence, we replaced all node features by random ones.

	Graph classification				*Node classification*
METHOD	DD	ENZYMES	MNIST	PROTEINS	Pattern
GCN-4	68.0 ± 3.6	22.0 ± 3.3	76.2 ± 0.5	68.8 ± 2.8	85.5 ± 0.4
GCN-3-TOGL-1	**75.1 ± 2.1**	**30.3 ± 6.5**	**84.8 ± 0.4**	**73.8 ± 4.3**	**86.6 ± 0.1**
GIN-4	75.6 ± 2.8	21.3 ± 6.5	83.4 ± 0.9	**74.6 ± 3.1**	84.8 ± 0.0
GIN-3-TOGL-1	**76.2 ± 2.4**	**23.7 ± 6.9**	**84.4 ± 1.1**	73.9 ± 4.9	**86.7 ± 0.1**
GAT-4	63.3 ± 3.7	21.7 ± 2.9	63.2 ± 10.4	67.5 ± 2.6	**73.1 ± 1.9**
GAT-3-TOGL-1	**75.7 ± 2.1**	**23.5 ± 6.1**	**77.2 ± 10.5**	**72.4 ± 4.6**	59.6 ± 3.3

Classifying benchmark data sets

While we improve baseline classification performance, the best performance is *not* driven by the availability of topological structures!

	Graph classification							*Node classification*
METHOD	CIFAR-10	DD	ENZYMES	MNIST	PROTEINS-full	IMDB-B	REDDIT-B	CLUSTER
GATED-GCN-4	**67.3 ± 0.3**	72.9 ± 2.1	65.7 ± 4.9	**97.3 ± 0.1**	**76.4 ± 2.9**	—	—	**60.4 ± 0.4**
WL	—	77.7 ± 2.0	54.3 ± 0.9	—	73.1 ± 0.5	71.2 ± 0.5	78.0 ± 0.6	—
WL-OA	—	**77.8 ± 1.2**	58.9 ± 0.9	—	73.5 ± 0.9	74.0 ± 0.7	87.6 ± 0.3	—
GCN-4	54.2 ± 1.5	72.8 ± 4.1	**65.8 ± 4.6**	90.0 ± 0.3	76.1 ± 2.4	68.6 ± 4.9	**92.8 ± 1.7**	57.0 ± 0.9
GCN-3-TOGL-1	61.7 ± 1.0	73.2 ± 4.7	53.0 ± 9.2	95.5 ± 0.2	76.0 ± 3.9	72.0 ± 2.3	89.4 ± 2.2	60.4 ± 0.2
	7.5	0.4	−12.8	5.5	−0.1	3.4	−3.4	3.4
GIN-4	54.8 ± 1.4	70.8 ± 3.8	50.0 ± 12.3	96.1 ± 0.3	72.3 ± 3.3	72.8 ± 2.5	81.7 ± 6.9	58.5 ± 0.1
GIN-3-TOGL-1	61.3 ± 0.4	75.2 ± 4.2	43.8 ± 7.9	96.1 ± 0.1	73.6 ± 4.8	**74.2 ± 4.2**	89.7 ± 2.5	60.4 ± 0.2
	6.5	4.4	−6.2	0.0	1.3	1.4	8.0	1.9
GAT-4	57.4 ± 0.6	71.1 ± 3.1	26.8 ± 4.1	94.1 ± 0.3	71.3 ± 5.4	73.2 ± 4.1	44.2 ± 6.6	56.6 ± 0.4
GAT-3-TOGL-1	63.9 ± 1.2	73.7 ± 2.9	51.5 ± 7.3	95.9 ± 0.3	75.2 ± 3.9	70.8 ± 8.0	89.5 ± 8.7	58.4 ± 3.7
	6.5	2.6	24.7	1.8	3.9	−2.4	45.3	1.8

Conclusion

- ‘If all you have is nails, everything looks like a hammer.’[4] Our data sets may actually *stymie* progress in GNN research because their classification does not necessarily require structural information.
- Nevertheless, higher-order structures (such as cliques) can be crucial in discerning between different graphs or data sets.
- Can we also learn sparse filtrations?

❤ **Acknowledgements**

My co-authors Edward, Karsten, Max, Michael, and Yves.

Software

`https://github.com/aidos-lab/pytorch-topological`
Looking for additional contributors!

[4] Credit: Mikael Vejdemo-Johannson