Link Prediction with Graph Deep Learning using Hodge Laplacians

GDL 2023
Group id: 01
Project id: SimComplexes

Qianbo Zang, Xinliang Song, Volodymyr Kyrylov
{qianbo.zang, xinliang.song, volodymyr.kyrylov}@usi.ch

Abstract

We present a simple framework SimGAE, which focus on link prediction by learning Node and Edge Embeddings with not only graph Laplacian (adjacent matrix) but also higher order graph Hodge Laplacian from simplices in a transductive setting. In addition to a common graph convolution with node to node message passing via edges we employ to edge to edge message passing via edges. Compared to other models, our model is much simpler without loss of performance. We even achieve some better performance on open datasets.

1 Introduction

Graphs are a fundamental way of modeling complex systems and are used in various fields such as social network analysis, recommendation systems, and biology. Link Property Prediction, the task of predicting missing or future connections between nodes in a graph, is a crucial problem in graph analysis with many applications.

Graph Deep Learning (GDL) has made significant progress in recent years in solving Node property prediction problems in graphs. In the Open Graph Benchmark (OBG), most datasets have achieved a Validation Accuracy of over 90%. However, despite this success, GDL still faces significant challenges in predicting link properties in graphs. For example, in the ogbl-ddi dataset, the best accuracy of the validation dataset is only 0.8258, indicating that there is still a long way to go in this field [1]. The bottleneck in GDL's performance on link property prediction problems is mainly due to the difficulty of capturing high-order structural information.

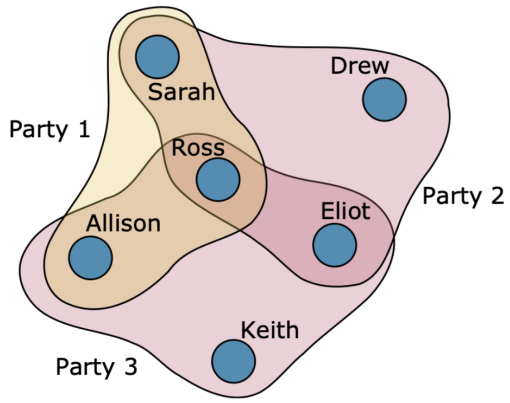

Figure 1. Sarah, Ross and Allison have a stronger relationship than pairwise relationships. These kind of relationships can be captured by hyper-edges.

Graph deep learning architectures build on encoded dependencies between samples using adjacency matrices. Convolution and attention operations use connectivity information from those adjacency matrix, aggregating or exchanging information from neighboring nodes via edges. However, some datasets contain more information between interactions of groups of nodes. We explore modeling these interactions using Hodge-Laplacian from simplicial complexes which play an essential role in detecting high-order information relationships going beyond lines (nodes connected via edges) to triangles connected by lines and tetrahedrons connected by triangles.

Graph Auto-Encoders(GAEs) [2] are end-to-end trainable neural network models for unsupervised learning, clustering and link prediction on graphs. Specially GAEs have successfully been used for Link prediction in large-scale relational data [3], which is trained to minimize the reconstruction loss between the real adjacency matrix and the decoded adjacency matrix.

In this work, we follow Y. Chen's [4] research by performing additional ablations, making architecture changes and using new datasets.

2 Related works

2.1 Link prediction

Link prediction, a well-established problem in scholarly literature, particularly within the field of graph network analysis, exhibits extensive applicability across domains such as social, biological, and transportation networks[5, 6, 7]. This problem is typically addressed by initially acquiring node embeddings, which effectively encode both graph topology and node characteristics within a lower-dimensional latent space. Notably, graph autoencoders represent a popular approach for learning node embeddings [2].

2.2 GAEs

Graph Autoencoder(GAE) [2] is an associated algorithm of deep learning in graph data. It encodes graph into a latent feature space and then reconstructs them from it. Firstly, the encoder generates network embeddings for each node by processing the graph with convolutional layers. Through

training the model, this representation efficiently captures the topological information of nodes. Next, negative links are randomly added to the original graph. This allows the model to perform binary classification on the positive links of the original edges and the negative links that were added. Subsequently, the decoder employs node embeddings to predict links for all edges, including the negative links. Finally, the decoder utilizes these computations to reconstruct the adjacency matrix of the image. The training objective of GAE is to minimize the reconstruction loss between the real adjacency matrix and the decoded adjacency matrix. By generating node embeddings and utilizing them for edge prediction, GAE offers an effective way to handle complex graph data.

2.3 Simplical network

The Graph Convolution Networks combined with Simplicial Complexes is an emerging trend in graph representation learning. Several recent studies have started to use Simplicial Complexes to carry out neural networks on graphs. For example, Ebli and his teammates [8] introduced a model known as Simplicial Neural Networks (SNNs), incorporating the simplicial Laplacian of a simplicial complex into a multi-layer perception framework. Chen introduced BScNets [4], incorporating SNNs and GAEs for link prediction.

3 Methodology

3.1 Higher Order Laplacian

The graph Laplacian is defined as the difference of diagonal graph in-degree matrix and the symmetric positive-definite adjacency matrix:

$$L = D - A$$

Repeated application of this matrix performs information diffusion in the graph via edges. We can generalize the notion of nodes to simplices: a single point is a 0-simplex, a line edge is a 1-simplex and a 2-simplex is a triangle. A k-simplex is defined by n+1 points, and a convex hull of of simplex points is a face. Thus, a 0-face is a node, a 1-face is an edge and a 2-face just a face. Fixing orientation of points in a k-simplex allows us to define a vector space C^k where bases are oriented k-simplices. We can define a boundary operator on this vector space that allows us to *move* between orders of simplices:

$$\partial_k : C^k \to C^{k-1}$$

A matrix representation of the boundary operator is denoted B_k. We can use B_1 and its adjoint to reconstruct the 0-Laplacian, capturing how information travels from node to node via edges:

$$L_0 = L_0^{up} = B_1 B_1^T = D - A$$

We can also send information between edges via nodes by using the *edge Laplacian* via B_1:

$$L_1^{down} = B_1^T B_1$$

B_2 is an edge to face incidence matrix, and it can be used to define L_1^{up}:

$$L_1^{up} = B_2 B_2^T$$

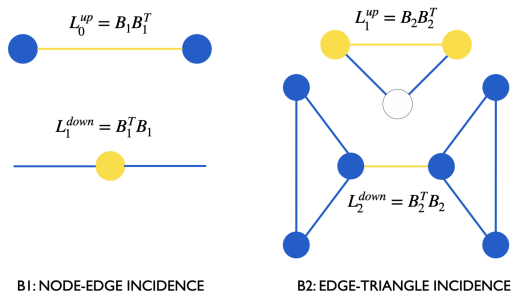

Figure 2. L_0^{up} can be considered that information flow from node to node via one edge. L_1^{down} can be considered that information flow from edge to edge via one node. L_1^{up} can be considered that information flow from edge to edge via one triangle. L_2^{down} can be considered that information flow from triangle to triangle via one edge

Exchanging information between the $k+1$ simplex level and the k-th level allows to define a so-called k-Hodge Laplacian:

$$L_k = B_k^T B_k + B_{k+1} B_{k+1}^T$$

Now, we can exchange information between triangles (2-simplices) in a graph just by performing graph learning operators on higher order Laplacians.

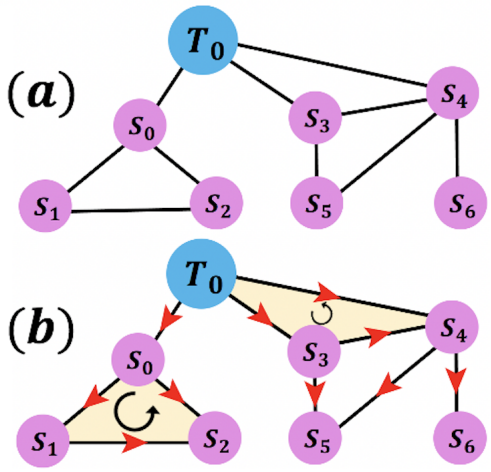

Figure 3. An example of simplicial complex from a classroom environment network, where T_0 is the teacher and S_i is the student_i ($i \in \{0, \ldots, 6\}$). (a) Graph structure of the classroom environment network; (b) Simplicial complex of the classroom environment network.

For clarity, we give an example of corresponding simplicial complex in a simple graph, which is shown in Figure 3. This figure is cited from BScNets, while they make an error when computing $B2$.

The node T_0 represents the only teacher in this graph and node S_i represents a student. Then each node represent a $0-$simplices. Each edge represent a $1-$simplices, each node of an edge is a face of this edge. Each triangle represent a $2-$simplices and each edge of a triangle is a face of this

triagnle.

There are 8 $0-$simplices in Figure 3, i.e., the seven nodes T_0 and S_0 to S_6. There are 10 $1-$simplices in Figure 3, i.e., the 10 edges. There are 3 $2-$simplices in Figure 3, i.e., the 3 triganles. , i.e., $\{S_0, S_1, S_2\}$, $\{T_0, S_3, S_4\}$ and $\{S3, S_4, S_5\}$.

Note that in order to represent simplices in (a) of Figure 3, we need to define the order of each node and then we can define the faces direction and the boundary matrices, as show in (b) of Figure 3.

As shown in Table 1 and 2, we give the the boundary matrices B_1 and B_2, i.e., the corresponding node-to-edge and edge-to-face incidence matrices of the graph in Figure 1 respectively. Then we can use these two matrices to compute L_0 and L_1.

Table 1: Boundary map B_1, i.e, node-to-edge incidence matrices of Figure 1.

	(T_0,S_0)	(T_0,S_3)	(T_0,S_4)	(S_0,S_1)	(S_0,S_2)	(S_1,S_2)	(S_3,S_4)	(S_3,S_5)	(S_4,S_5)	(S_4,S_6)
T_0	-1	-1	-1	0	0	0	0	0	0	0
S_0	0	0	0	-1	-1	0	0	0	0	0
S_1	0	0	0	1	0	-1	0	0	0	0
S_2	0	0	0	0	1	1	0	0	0	0
S_3	0	0	0	0	0	0	-1	-1	0	0
S_4	0	0	0	0	0	0	1	0	-1	0
S_5	0	0	0	0	0	0	0	1	1	0
S_6	0	0	0	0	0	0	0	0	0	1

Table 2: Boundary map B_2, i.e, edge-to-triangle incidence matrices of Figure 1.

	(T_0,S_3,S_4)	(S_0,S_1,S_2)	(S_3,S_4,S_5)
(T_0,S_0)	0	0	0
(T_0,S_3)	1	0	0
(T_0,S_4)	-1	0	0
(S_0,S_1)	0	1	0
(S_0,S_2)	0	-1	0
(S_1,S_2)	0	1	0
(S_3,S_4)	1	0	1
(S_3,S_5)	0	0	-1
(S_4,S_5)	0	0	1
(S_4,S_6)	0	0	0

While Hodge Laplacians come from cohomology and might sound intimidating to grasp, [9] introduces us to the topic merely requiring working linear algebra and graph theory (and a lot of attention). Alluded definition of B_2 hints that we need to count triangles. Typically, our graph far differs from a fully connected graph, thus the adjacency matrix is a sparse matrix. The operation for counting triangles in a graph often requires $O(|E|^2)$ time, we will use the sources in [4] with efficient algorithms to compute $B_{1,2}$ matrices that we will be employing on datasets with simplicial structure.

3.2 SimGAE

Now we can introduce our model, which graph atuencoders with simplified simplices' laplacians, which is aslo a simplified model of BScNets.

We first give the Quasi Hodge Laplacians(QHL) that used in our model:

$$\mathfrak{L}_1 = \left[\begin{array}{c|c} \boldsymbol{L}_0 & O \\ \hline O & \boldsymbol{L}_1 \end{array}\right]. \tag{1}$$

$\mathfrak{L}_1$ is composed of the $0-$th and $1-$th *normalized* Hodge Laplacians $\boldsymbol{L}_0$ and $\boldsymbol{L}_1$. We left the elements in the two off-diagonal are all 0. Of course, the Quasi Hodge Laplacians can incorporate higher order Hodge Laplacians and we are still implementing it. Thus we use the $2-$order Quasi Hodge Laplacians currently and will update once we successfully implement higher order version.

Quasi Hodge Laplacians Convolution Following how GCN is constructed with graph Laplacian and learn he distance of two node embeddings, we construct the Hodge-style convolution with our Quasi Hodge Laplacians operator. We adopt our operator $\mathfrak{L}_1$ to capture higher-order information of the graph. The Quasi Hodge Laplacians Convolution(QHC) is given by

$$\boldsymbol{Z}^{(\ell+1)} = \left(\mathfrak{L}\boldsymbol{Z}^{(\ell)}\boldsymbol{\Theta}_1^{(\ell)}\right)\boldsymbol{\Theta}_2^{(\ell)}, \tag{2}$$

where $\boldsymbol{Z}^{(0)} = \boldsymbol{X} \in \mathbb{R}^{n\times d}$ is node features matrix, $\boldsymbol{\Theta}_1^{(\ell)} \in \mathbb{R}^{d\times(d_0+d_1)}$ and $\boldsymbol{\Theta}_2^{(\ell)} \in \mathbb{R}^{(d_0+d_1)\times d_{\text{out}}}$ are two trainable weight matrices at layer ℓ, and d_0, d_1 denotes the dimension of $\boldsymbol{L}_0$ and $\boldsymbol{L}_1$ respectively, and d_{out} denotes the dimension of node embedding at the (ℓ)-th layer through the QHC operation. For the link prediction task, we use the Fermi-Dirac decoder [10] to compute the distance between the two nodes. Formally,

$$\text{dist}_{uv}^{\text{QHC}} = (\boldsymbol{Z}_u^{(\ell+1)} - \boldsymbol{Z}_v^{(\ell+1)})^2, \tag{3}$$

where $\text{dist}^{\text{QHC}}{}_{uv} \in \mathbb{R}^{1\times d_{\text{out}}}$ is the distance between nodes u and v in a local spatial domain.

Graph Convolution Layer Similar to the process of computing distances between learnable node embeddings with the Hodge-style adaptive block convolution, we also use graph convolution operation to capture geographic information equipped with node attributes and compute the distance of the node embeddings, i.e,

$$\boldsymbol{H}^{(\ell+1)} = \tilde{\boldsymbol{L}}\boldsymbol{H}^{(\ell)}\boldsymbol{\Theta}_3^{(\ell)}, \tag{4}$$

$$\text{dist}_{uv}^{\text{GC}} = (\boldsymbol{H}_u^{(\ell+1)} - \boldsymbol{H}_v^{(\ell+1)})^2, \tag{5}$$

where $\tilde{\boldsymbol{L}} = \boldsymbol{D}_v^{-1/2}(\boldsymbol{A}+\boldsymbol{I})\boldsymbol{D}_v^{-1/2}$ (where $\boldsymbol{D}_v$ is the degree matrix of $\boldsymbol{A}+\boldsymbol{I}$, i.e., $[\boldsymbol{D}_v]_{ii} = \sum_j [\boldsymbol{A}+\boldsymbol{I}]_{ij}$), $\boldsymbol{H}^{(0)} = \mathbf{X} \in \mathbb{R}^{n\times d}$ is node features matrix, $\boldsymbol{\Theta}_3^{(\ell)} \in \mathbb{R}^{d\times d_{\text{out}'}}$ is the trainable weight matrix, $d_{\text{out}'}$ denotes the dimension of node embedding at the (ℓ)-th layer through the graph convolution operation, $\text{dist}_{uv}^{\text{GC}} \in \mathbb{R}^{1\times d_{\text{out}'}}$ is the distance between nodes u and v in global spatial domain.

Distance between Two Nodes Now we can combine the distances from our Hodge-style convolution QHC (3) and original graph convolution (5) outputs into Multi-layer Perceptron (MLP)

$$\text{dist} = \text{ReLU}(f_{\text{MLP}}([\pi_\alpha \times \text{dist}^{\text{GC}}, \pi_\beta \times \text{dist}^{\text{QHC}}])),$$

where $[\cdot,\cdot]$ denotes the concatenation of the outputs from QHC and original graph convolution. π_α and π_β are two hyperparameters, which represent the weight of each distance when we combine QHC distance and GC

distance respectively. f_{MLP} is an MLP layer that maps the concatenated distance embedding to a d_o-dimensional vector. Then we can use distance to represent the possibility of the existence of an edge.

3.3 The overall framework of SimGAE

Now we can give an overall framework of our Sim GAE model.

1. the ordinary graph convolution(GC) captures node embeddings by aggregating the embeddings of neighbors using graph Laplacians,
2. Quasi Hodge Laplacians Convolution(QHC) captures node embeddings by aggregating the embeddings of neighbors using graph Hodge Laplacians. It can captures information from (1)message passing from edges to edges through nodes, (2) massage passing from edges to edges through triangles.
3. we concatenate two GC and QHC embeddings into a MLP to predict whether an edge exists between any pair of nodes.

3.4 Compared to BScNets

In BScNets, Chen consider to modify linear operator $\mathfrak{L}_1$ such that it describe interaction relations among $\boldsymbol{L}_0$ and $\boldsymbol{L}_1$. However from our experiments, we find such modifications on the operator $\mathfrak{L}_1$ even reuslts in worse performance. This is the main diffrence between SimGAE and BScNets.

Moreover, we find the actual model used in BScNets' source code is a little bit diffrent from the expression in their paper. They use $L_1^{down} = B_1^T B_1$ to construct $\boldsymbol{L}_0$ in their code, while the mode in paper $\boldsymbol{L}_0$ is the real L_0.

4 Implementation of SimGAE

4.1 Datasets

We use 3 open datasets to implement our experiments: Cora, PubMed and Disease. We are also working on OBG-ddi dataset.

4.2 Experimental setup and computational requirements

We perform experiments on our private server. The experimental setup was made in Python Framework with PyG official libraries as examples and support documents. The following versions were applied: PyTorch 2.0.0, Python 3.8(ubuntu20.04), Cuda 11.8. We exploit the available GPU from RTX 4090' and CPU is Xeon(R) Platinum 8375C @ 2.90GHz. It cost less than 1 second for 10 epochs for *cora* dataset and can early stop within 2000 epochs. As for ogb-ddi dataset, we used a single RTX 3090 GPU and a single experiment takes under one hour.

The original code of BScNets is available at `https://github.com/proger/BScNets`. Although our implementation is based on it, we detect some bugs of their code and modify our neural network structure. Moreover we improve the readability of the source code, as the source code has very few comments.

4.3 Hyperparameters

The hyperparameters are the same as in [4] except for $weight\_decay$.

Table 1. Hyperparameters Setting

Hyperparameter	Setting
`Validation proportion`	0.05
`Testing proportion`	0.003, 0.001
`activation function`	`Relu` & `Sigmoid`
`dropout probability`	0.5
`edge sampling numbers`	2500
`learning rate`	0.005
`edge sampling numbers`	1e-4
π_α	1
π_β	0.1

.

4.4 Results

As shown in Table 2, we compared our SimGAE with other GNN models. Other experiments results are cited from [4]. We can see that although we have simplified the BScNETs, we even have a little bit performance improvement on Cora, PubMed and Disease datasets.

Table 2. ROC AUC and standard deviations for link prediction. Bold numbers denote the best results. * means statistically significant result.

Model	Cora	PubMed	Disease
MLP	83.15 ± 0.51	84.10 ± 0.97	72.62 ± 0.61
HNN [11]	89.00 ± 0.10	94.87 ± 0.11	75.10 ± 0.35
GCN [12]	90.42 ± 0.28	91.11 ± 0.55	64.70 ± 0.56
GAT [13]	93.89 ± 0.13	91.22 ± 0.12	69.99 ± 0.32
SAGE [14]	86.24 ± 0.65	85.96 ± 1.16	65.91 ± 0.33
SGC [15]	91.67 ± 0.20	94.10 ± 0.20	65.21 ± 0.23
SEAL [5]	92.55 ± 0.50	92.42 ± 0.12	85.23 ± 0.79
HGCN [7]	93.00 ± 0.45	96.29 ± 0.18	90.80 ± 0.30
PEGN [6]	93.13 ± 0.50	95.82 ± 0.20	83.61 ± 1.26
TLC-GNN [16]	94.22 ± 0.78	97.03 ± 0.10	86.19 ± 1.23
BScNets [4]	$^*\mathbf{94.90 \pm 0.70}$	$^*\mathbf{97.55 \pm 0.12}$	$^*\mathbf{98.60 \pm 0.58}$
ours	$^*\mathbf{95.97 \pm 0.41}$	$^*\mathbf{97.88 \pm 0.30}$	$^*\mathbf{98.79 \pm 0.79}$

We extend the implementation to experiment on DrugBank 5.0 [17] also known as *ogbl-ddi*. It is a drug-drug interaction network where the task is to predict interactions unobserved in training. The graph has 4367 nodes and 1334889 recorded node interactions. As a baseline implementation we're using GraphSAGE with code provided by OGB team available at `https://github.com/snap-stanford/ogb`.

Table 3. ogbl-ddi Test

Methods	AP	AUC-ROC	Hits@20
`GCN+GraphSAGE`	–	–	0.56
`BScNet_1`	0.92	0.92	< 0.01

4.5 Discussion

Our experimental performance are better than the traditioanl model without considering simplices and Hodge Laplacians in the graph. This improvement shows that high-order graph information is important for link prediction and Hodge Laplacians captures the higg order information in the graph

very well. Our SimGAE is actually a simplified variant of BscNets. But we found that although some theoretical support of BscNets was lost, SimGAE perform better than bscnets in some datasets. This may be because BScNets may lost some simplices information when building Hodge style convolution.

Unlike other datasets where average precision and ROC are reported, we could not produce a result with a competitive Hits@20 value. Hits@20 counts the number of correct hits within top-20 ranked results for each node. We will be investigating the cause of this model behavior and suspect it might be due to scale. This dataset has 1000 times more node interactions than Cora and it might be fruitful to investigate other edge sampling strategies.

5 Conclusion

Learning with higher-order Laplacians is a very powerful technique that comes with scalability issues: computing B_2 exactly given a graph is an expensive operation. Simplifying the problem by subsampling edges doesn't seem to work on all datasets. We are determined to explore solutions to scalability problem in the future work.

5.1 Future Work

In this report, we only considered 2-order Hodge Laplacians, i.e, we only worked on 2-simplices. In the datasets with graphs that have stronger connectivity, there may exist many 3 or 4 simplices in the graph. Thus Quasi Hodge Laplacians with higher order may have much better performance.

In this report, we just use the original GAEs for node embedding. Thus an other possible improvement is that we could consider more advanced embedding methods.

The main drawback of SimGAE is that we only worked in tranducive setting, which means that it is not suitable for very large graph or dynamic graph. An other possible improvement is that we try to figure out how Hodge Laplacians can be implicitly represented in inductive graph network such as GraphSage. Or we also could consider how to capture triangles' information when we sample and aggregate neighbour's' information.

References

[1] Zixiao Wang, Yuluo Guo, Jin Zhao, Yu Zhang, Hui Yu, Xiaofei Liao, Hai Jin, Biao Wang, and Ting Yu. Gidn: A lightweight graph inception diffusion network for high-efficient link prediction. *arXiv preprint arXiv:2210.01301*, 2022.

[2] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv:1611.07308*, 2016.

[3] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *The Semantic Web: 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3–7, 2018, Proceedings 15*, pages 593–607. Springer, 2018.

[4] Yuzhou Chen, Yulia R. Gel, and H. Vincent Poor. Bscnets: Block simplicial complex neural networks. In *AAAI Conference on Artificial Intelligence*, 2021.

[5] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In *NeurIPS*, volume 31, pages 5165–5175, 2018.

[6] Qi Zhao, Ze Ye, Chao Chen, and Yusu Wang. Persistence enhanced graph neural network. In *AISTATS*, pages 2896–2906, 2020.

[7] Ines Chami, Zhitao Ying, Christopher Ré, and Jure Leskovec. Hyperbolic graph convolutional neural networks. In *NeurIPS*, volume 32, pages 4868–4879, 2019.

[8] Stefania Ebli, Michaël Defferrard, and Gard Spreemann. Simplicial neural networks. In *NeurIPS 2020 Workshop on TDA and Beyond*, 2020.

[9] Lek-Heng Lim. Hodge laplacians on graphs. *ArXiv*, abs/1507.05379, 2015.

[10] Maximillian Nickel and Douwe Kiela. Poincaré embeddings for learning hierarchical representations. In *NeurIPS*, volume 30, pages 6338–6347, 2017.

[11] Octavian-Eugen Ganea, Gary Bécigneul, and Thomas Hofmann. Hyperbolic neural networks. In *NeurIPS*, pages 5350–5360, 2018.

[12] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

[13] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

[14] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, pages 1025–1035, 2017.

[15] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *ICML*, pages 6861–6871, 2019.

[16] Zuoyu Yan, Tengfei Ma, Liangcai Gao, Zhi Tang, and Chao Chen. Link prediction with persistent homology: An interactive view. In *ICML*, pages 11659–11669, 2021.

[17] David Scott Wishart, Yannick Djoumbou Feunang, Anchi Guo, Elvis J. Lo, Ana Marcu, Jason R. Grant, Tanvir Sajed, Daniel Johnson, Carin Li, Zinat Sayeeda, Nazanin Assempour, Ithayavani Iynkkaran, Yifeng Liu, Adam Maciejewski, Nicola Gale, Alex Wilson, Lucy Chin, Ryan Cummings, Diana Le, Allison Pon, Craig K. Knox, and Michael Wilson. Drugbank 5.0: a major update to the drugbank database for 2018. *Nucleic Acids Research*, 46:D1074 – D1082, 2017.