Continual Graph Learning

XIKUN ZHANG

Supervisor: Prof. Dacheng Tao
Auxiliary Supervisor: Dr. Baosheng Yu

A thesis submitted in fulfilment of
the requirements for the degree of
Doctor of Philosophy

This research reported in this thesis was supported by
the award of a Research Training Program scholarship
to the PhD Candidate.

School of Computer Science
Faculty of Engineering
The University of Sydney
Australia

March 2024

Statement of Originality

This is to certify that to the best of my knowledge, the content of this thesis is my own work. This thesis has not been submitted for any degree or other purposes.

I certify that the intellectual content of this thesis is the product of my own work and that all the assistance received in preparing this thesis and sources have been acknowledged.

Xikun Zhang
School of Computer Science
Faculty of Engineering
The University of Sydney

Authorship Attribution Statement

This thesis contains several chapters that were previously published by Xikun Zhang as the first author in the following publications:

(1) **Xikun Zhang**, Dongjin Song, and Dacheng Tao. Cglb: Benchmark tasks for continual graph learning. In: *Advances in Neural Information Processing Systems 35, 13006-13021.* Presented in Chapter 3. I designed the research, implemented the systems, collected the data, conducted the experiments, and wrote the draft of the paper.

(2) **Xikun Zhang**, Dongjin Song, and Dacheng Tao. Hierarchical prototype networks for continual graph representation learning. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence 45 (4), 4622-4636.* Presented in Chapter 4. I designed the research, implemented the systems, conducted the experiments, and wrote the draft of the paper.

(3) **Xikun Zhang**, Dongjin Song, and Dacheng Tao. Sparsified Subgraph Memory for Continual Graph Representation Learning. In: *2022 IEEE International Conference on Data Mining (ICDM), 1335-1340.* Presented in Chapter 5. I designed the research, implemented the systems, conducted the experiments, and wrote the draft of the paper.

(4) **Xikun Zhang**, Dongjin Song, and Dacheng Tao. Ricci Curvature-Based Graph Sparsification for Continual Graph Representation Learning. In: *IEEE Transactions on Neural Networks and Learning Systems.* Presented in Chapter 6. I designed the research, implemented the systems, conducted the experiments, and wrote the draft of the paper.

In addition to the statements above, in cases where I am not the corresponding author of a published item, permission to include the published material has been granted by the corresponding author.

Xikun Zhang	Date

As supervisor for the candidature upon which this thesis is based, I can confirm that the authorship attribution statements above are correct.

Dacheng Tao	Date

Acknowledgements

I want to start by thanking Prof. Dacheng Tao for his invaluable guidance throughout my PhD journey. His mentorship has been crucial in my transformation from a beginner to a researcher. Prof. Tao's passion for research and dedication to excellence have deeply inspired me. I am grateful for his teachings, which will remain with me for a lifetime.

I also extend my heartfelt thanks to Prof. Dongjin Song. His support, kindness, and enlightening discussions have greatly contributed to my academic journey. His insights into academic writing and research have been particularly impactful. I am deeply appreciative of his guidance and support.

Lastly, I extend my heartfelt thanks to my family, friends, and all who love me and whom I love. Their support, both physical and spiritual, has been a constant source of strength throughout my life.

Abstract

Various real-world graphs grow with time, necessitating the research of continual graph learning (CGL), which aims to accommodate new tasks over newly emerged graph data while maintaining the model performance over existing tasks. The works of this thesis are twofold.

First, we systematically study the CGL task configurations in different application scenarios and develop a comprehensive Continual Graph Learning Benchmark (CGLB) curated from different public datasets. CGLB contains comprehensive CGL tasks under various experimental settings, as well as a toolkit for developing CGL techniques. With CGLB, the barrier to exploring CGL is greatly lowered and researchers can focus on the model development without worrying about tedious work on dataset pre-processing or unseen pitfalls.

Second, we developed a series of CGL techniques: 1) Hierarchical Prototype Networks (HPNs), 2) Sparsified Subgraph Memory (SSM), and 3) Subgraph Episodic Memory (SEM). Hierarchical Prototype Networks (HPNs) is designed to extract basic shareable features and store them into prototypes. In this way, the forgetting problem can be alleviated by knowledge sharing and independently updated prototypes. Next, SSM is a memory-replay based CGL technique, which stores a set of representative historical data from previous tasks to replay while learning new tasks. While topological information is critical in characterizing graph data, existing memory replay based CGL techniques only store individual nodes for replay and do not consider the topological information due to the memory explosion problem. To this end, SSM is designed to sparsify the selected computation graphs into fixed size before storing them into the memory. In this way, we can significantly reduce the memory consumption of a computation subgraph, and for the first time enable GNNs to utilize the explicit topological information for memory replay. Based on SSM, we developed the SEM, which adopts graph Ricci-curvature as the criteria during the computation subgraph sparsification.

Finally, in experiments, we study various real-world graph data including social network, citation network, product co-purchasing network, scene graph, and molecule graphs.

Contents

CHAPTER 1

Introduction

1.1 Background and Motivation

In real-world applications, many graph data are generated continuously. For instance, in a citation network, new types of papers and their associated citations may emerge and an ideal document classifier needs to continuously adapt its parameters to distinguish the documents of newly emerged research fields [57, 122, 129]; for drug design, new categories or properties of molecules may be encountered and a molecule property prediction model will need to keep updating its parameters to learn new molecule categories or properties [57]. When the new data follow different distributions from the previously observed data, updating the model parameters to accommodate new data will cause the old knowledge of the previous data to be overwritten. In other words, the model will experience the *catastrophic forgetting* phenomenon, which means that the model's performance on previous tasks decreases drastically after learning new tasks. Therefore, given either a continuously expanding graph (with new types of nodes and associated edges) or new categories of graphs continuously emerging, for practical concerns, it is important to develop Continual Graph Learning (CGL) that can accommodate new tasks over newly emerged graph data while maintaining the model performance over existing tasks.

1.2 Thesis Outline

The works of this thesis include both studies on CGL benchmark construction and technical studies focusing on proposing novel and effective CGL techniques.

First, unlike continual learning on Euclidean data (*e.g.*, images, texts, etc.) that has established benchmarks and unified experimental settings [56, 72, 88, 118, 37, 74, 47, 33], CGL is immature and with very rare public benchmark tasks being available. Moreover, due to the variety and complexity of graph data, different datasets have been adopted by existing works with different experimental settings. This causes great difficulties in model comparisons and hinders the development of this field. Targeting this challenge, in our developed Continual Graph Learning Benchmark (CGLB) [121], we provide a systematic taxonomy of different CGL scenarios that include (1) node-level CGL (N-CGL), which deals with node-level prediction on a single growing graph with new types of nodes continuously emerging; and (2) graph-level CGL (G-CGL), which deals with graph-level prediction with new categories of graphs continuously appear. Moreover, we clarify the difference between the currently widely adopted task-incremental learning (task-IL) setting and the underexplored yet more challenging class-incremental learning (class-IL) setting, and point out the proper application scenarios of these two settings. Based on the established taxonomy, we construct the benchmark tasks of CGLB for each setting. Along with CGLB, we provide a comprehensive toolkit for training, evaluation, and visualization of different CGL models, which could greatly alleviate the burden/risks to investigate CGL problems. Finally, with CGLB and the toolkit, we conduct comprehensive studies on the existing state-of-the-art methods, which reveal the effectiveness of these methods in different scenarios and point out challenges of the existing models under certain settings. With all these efforts, we not only lower the barrier to explore the CGL problem but also provide a systematic overview of the current situation of CGL and point out several challenging directions for future studies. At the end of the study of CGLB, we also extend our scope to scene graph related tasks, and justified the importance of continual learning for scene graph research.

Second, we also made substantial technical contribution from various perspectives. The technical works include Hierarchical Prototype Networks (HPNs) [122], sparsified subgraph memory (SSM) [124], and the Ricci-curvature based subgraph episodic memory (SEM) [123].

HPNs is a completely novel framework, which is designed to continuously extract different levels of abstract knowledge (in the form of prototypes) from graph data such that new

knowledge will be accommodated while earlier experience can still be well retained. Within this framework, representation learning is simultaneously conducted to avoid catastrophic forgetting, instead of considering these two objectives separately. Although learning independent prototypes for different tasks is appealing for preventing forgetting, it may incur unbounded memory consumption with an unlimited number of new tasks. Therefore, we propose to denote each node with a composition of basic prototypes so that nodes from different tasks and their associated relational structures are represented with compositions of a limited set of basic prototypes. Take the social network as an example, if a node (person) falls into certain category, it can be decomposed into basic atomic characteristics belonging to a set of attributes (*e.g.*, gender, nationality, hobby, *etc.*,) and the relationship between a pair of nodes can also be categorized into different basic types (*e.g.*, friends, coworkers, *etc.*). Inspired by these facts, we first develop the Atomic Feature Extractors (AFEs) to decompose each node into two sets of atomic embeddings, *i.e.*, atomic node embeddings which encode the node feature information and atomic structure embeddings which encode its relations to neighboring nodes within multi-hop. Next, we present Hierarchical Prototype Networks to adaptively select, compose, and store representative embeddings with three levels of prototypes, *i.e.*, atomic-level, node-level, and class-level. Given a new node, only the relevant AFEs and prototypes in each level will be activated and refined, while others are uninterrupted. The memory efficiency and the continual learning capability of the proposed HPNs are theoretically and experimentally validated.

Next, we introduce the SSM, which is a memory replay based technique. To utilize the topological information while maintaining a tractable space complexity at the same time, we propose the sparsified subgraph memory (SSM) [124] to sparsify a computation subgraph to a fixed size before storing it into the memory. Specifically, to sparsify the computation subgraph of a node v while maintaining the connectivity of sparsified subgraphs at the same time, we start by sampling the 1-hop neighbors, and then iteratively sample the higher-order neighbors hop by hop. In this way, since we can fix the number of nodes to sample at each hop, the size of the sparsified computation subgraph will be independent of the original computation subgraph, *i.e.*, the memory space complexity can be reduced from $\mathcal{O}(d^L)$ to $\mathcal{O}(1)$. In this

work, we adopted two sparsification strategies using uniform sampling and degree based sampling, which are simple yet empirically effective.

Following SSM, SEM is an advanced version of memory replay based CGL technique. GNNs generate graph representations by neighborhood aggregation, while different neighbors contribute differently to the obtained representations. Therefore, based on SSM, we further incorporate the importance of different neighbors when selecting which nodes to keep. Accordingly, we propose to sparsify the computation subgraph and preserve the most informative structures (nodes and edges) by utilizing the edge based Ricci curvature [86, 110]. Specifically, the Ricci curvature quantifies how easily information is propagated between two nodes. Given a pair of nodes u and v connected by an edge e_{uv}, the curvature Ric(u,v) is calculated based on the surrounding topological connections like the triangles and 4-cycles based at e_{uv}. Consequently, a higher curvature implies that information could be propagated more easily from u to v and vice versa (as shown in Figure 4.1(b)). In other words, among all edges connected to a node v, the ones with higher curvatures contribute more in generating the representation of v and should be preserved. In contrast, the edges with low curvatures provide little information to v and can be deleted without significant influence on the representation of v. Therefore, to sparsify the computation subgraph of a node v containing its L-hop neighbors, we propose to sample a subset of the edges and their associated nodes based on curvatures. The Ricci curvature has several formulations. In SEM, we adopt the Balanced Forman curvature [86] which is theoretically justified to be negatively correlated to the information bottleneck. Considering that the computation complexity of Balanced Forman curvature is $|\mathbb{E}|d_{\max}^2$, which is time-consuming to compute on dense graphs, we circumvent computing the exact curvature values by approximating it with the graph diffusion process formulated as a lazy random walk. We theoretically demonstrate that under certain rules, the probability distribution at a node u of a lazy random walk starting at node v is positively correlated to the Ricci curvature between v and u.

CHAPTER 2

Literature Review

This thesis is mainly related to: graph representation learning, continual graph learning, continual learning, graph sparsification, and graph curvature.

2.1 Graph Representation Learning

Graph representation learning aims to encode both the feature information from nodes and the topological structure of the incoming graph. Traditional methods relied on graph statistics or hand-crafted features [8, 53]. Recently, a great amount of attention has been paid to graph neural networks (GNNs), such as graph convolutional network (GCNs) [43], GraphSAGE [35], Graph Attention Networks (GATs) [87], and their extensions [105, 18, 7, 127, 125, 49, 30, 107, 120]. Instead of focusing on shallow networks, there are also works on building deep GCNs to further increase the capacity of GNNs [51, 19, 75, 126, 95]. Typically, graph representation learning models assume the graphs to be static, and will experience catastrophic forgetting problem when learning a sequence of tasks, which is shown in our experiments.

2.2 Continual Graph Learning

Recently, continual learning on graph data is also attracting increasingly more attention due to its enormous value in various practical scenarios. For example, a community detection model has to keep adapting to nodes from newly emerged communities in social networks, a document classifier needs to keep learning to distinguish documents of newly emerged research areas in citation networks, *etc*. To this end, several methods have been introduced

for continual graph learning [129, 14, 122, 57, 90, 106, 24, 124, 73, 36, 16, 46, 116, 70, 26, 11]. These methods include regularization based ones like topology-aware weight preserving (TWP) [57] which preserves crucial parameters and topological structures of the previous tasks via regularization terms, parametric isolation based approaches like HPNs [122] which adaptively select different parameter combinations for different tasks, and memory replay based methods like ER-GNN [129] which stores representative nodes from the previous tasks in a buffer which are replayed when learning new tasks, and SSM [124] that stores computation subgraphs sparsified via random sampling.

Finally, it is also worth noting the essential difference between continual graph representation learning, dynamic graph representation learning [29, 92, 36, 115, 68, 132, 61], and few-shot graph learning [130, 34, 109]. Dynamic graph representation learning focuses on capturing the temporal dynamics of nodes with all previous information being accessible. On the contrary, continual graph representation learning focuses on alleviating the forgetting of previous tasks, therefore the previous data is inaccessible when learning new tasks. Few-shot graph learning aims at fast adapting the model to new tasks, which adopts a completely different setting. During training, few-shot learning models have access to data of all tasks simultaneously, while models under the continual learning setting can only access the data of the current task and are not allowed to access previous data. During testing, few-shot learning models are evaluated on new tasks and need to be fine-tuned with the test data, while the continual learning models are evaluated over existing tasks without any new data for fine-tuning.

2.3 Continual Learning

Continual learning aims to overcome the well-known catastrophic forgetting problem that a model's performance on previous tasks decreases significantly after being trained on new tasks. Existing works for continual learning can be categorized as regularization-based methods, memory-replay based methods, and parametric isolation based methods. In the following, we give detailed introductions on these approaches, as well as their applicability to graph data.

Regularization-based methods penalize the model objectives to maintain satisfactory performance on previous tasks [42, 52, 44, 27, 77]. For instance, Li and Hoiem [52] introduced Learning without Forgetting (LwF) which uses knowledge distillation to constrain the shift of parameters for old tasks; Kirkpartick et al. [44] proposed elastic weight consolidation (EWC) that adds a quadratic penalty to prevent the model weights from shifting too much. Recent works [27, 77] seek to constrain the gradients for new tasks in a subspace orthogonal to the updating directions that are important for previous tasks. Although the regularization-based methods can alleviate the forgetting on previous tasks, the constraints on the model weights reduce a model's plasticity for new tasks, resulting in inferior performance compared to the other two approaches. Regularization-based methods can be directly applied to graph neural networks and are included as baselines in our experiments.

Memory-replay based methods constantly feed a model with representative data of previous tasks to prevent forgetting [59, 80, 5, 13, 22]. One example is Gradient Episodic Memory (GEM) [59] that stores representative data in episodic memory and adds a constraint to prevent the loss of the episodic memory from increasing and only allow it to decrease. Instead of storing data, Shin et al. [80] added a generative model to generate pseudo-data of previous tasks to be interleaved with new task data for rehearsal. Recent works also look for better designs of memory replay to facilitate continual learning agents [13, 22]. Memory-replay based methods are currently one of the most effective approaches for alleviating catastrophic forgetting, but these methods are not suitable for graph data, as they cannot store the topological information, which is crucial for representing graph data. In contrast, our proposed HPNs explicitly designed Atomic Feature Extractors (AFEs) to encode both node and topological information. Moreover, memory-replayed methods require rehearsal of old data each time when a new task is learned. This introduces extra computation burdens.

Parametric isolation based methods adaptively introduce new parameters for new tasks to avoid the parameters of previous tasks being drastically changed [76, 113, 112, 96, 100]. For instance, progressive network [76] allocates a new sub-network for each new task and block any modification on the previously learned networks. Yoon et al. [113] proposed a more flexible model (DEN) that dynamically adds new neurons to accommodate new tasks.

Recently, various innovative approaches to allocate separated parameters for different tasks have been developed [100, 112, 96]. Besides the concrete models, Knoblauch et al. [45] analyzed the required capability of an optimal continual learning agent. Although parametric isolation based methods are effective for alleviating the catastrophic forgetting problem, they also consistently increase the model complexity and memory consumption which could be a problem. Recently developed methods have used certain mechanisms to control memory consumption, such as merging parameters for similar tasks [112]. However, as task specific parameters are fitted to each individual task, these parameters are seldom reused.

2.4 Graph Sparsification

Graph sparsification has been extensively studied in the past few years [41, 64, 15, 17, 50, 63, 111, 60, 128]. However, unlike the curvature based sparsification proposed in our work, these methods are not specially developed to preserve the most informative neighbors for training GNNs. For instance, several prominent works have been presented to preserve certain predefined metrics or statistics on graphs [41, 64, 15, 17, 50, 63]. In addition, these approaches may also encounter high computational burdens. Recently, various GNN explanation [111, 60, 54, 114, 6, 81] or graph denoising [128] techniques have been developed to find the most informative structures. These methods, however, typically require training another network or iterative optimizations to explain one trained GNN, which will result in more resource consumption and are not suitable for continual learning. Despite this, some neighborhood sampling strategies could be adopted for our proposed SEM. For example, GraphSAGE [35] utilizes randomly sampled neighbors at each layer to reduce the computational burden. Similarly, DropEdge [75] randomly removes edges as a regularization.

2.5 Graph Curvature

The vast majority of GNNs adopt the message passing paradigm [32, 43, 102, 127, 2, 125, 99, 126], *i.e.* iteratively propagating the information of each node to its neighbors along the edges. Therefore, the information flow is largely determined by the topological structures

of the graph. To investigate how smoothly the information travels on graphs and detect the information bottlenecks, the Ricci curvature, which is calculated based on the topological structures, is adopted as a powerful tool [86, 110]. The concept of curvature originates in the research on differential geometry for studying manifolds and could be intuitively understood as a measurement of the shape deformation of a manifold compared to a *flat* circumstance. Similarly, the curvature on a graph the grid graphs, or the discrete Euclidean geometry, can be viewed as *flat* graphs without deformation, and the curvature is zero. In the complete graph (the first graph of Figure 4.1 (b)), or the discrete spherical geometry, the edges starting at two different nodes tend to converge and the curvature is positive. The third graph of Figure 4.1 (b) illustrates the discrete hyperbolic geometry where the walks starting at two nodes tend to diverge and the curvature is negative. The curvature we focus on in this work are defined based on edges, and the different shape deformation scenarios mentioned above also correspond to different hardness for information to flow along the edges. As its discrete version, graph curvature can be intuitively understood as a metric on the graph bottleneck. For example, given the nodes u and v in Figure 4.1 (b), with a positive curvature in a complete graph, there are multiple paths connecting them and the information can easily flow between u and v. While in the discrete hyperbolic scenario with negative curvature, there is no other paths for the information to flow between u and v except the edge e_{uv}. The Ricci curvature has different formulations. The Forman-Ricci curvature [28, 82] is defined based on the combinatorial properties of the graph with a relatively low computation complexity. But its definition is biased to the negative curvatures. The Ollivier-Ricci curvature [55] on two nodes is defined based on the difference between their Wasserstein distance and shortest path distance. Its local quantities are hard to control and it has a higher computation complexity. Targeting these drawbacks, the Balanced Forman curvature [86] is proposed and is theoretically proven to be negatively correlated to the information bottlenecks when applying MPNNs on graph data. Therefore, we also adopt the Balanced Forman curvature to detect the most informative edges for sparsifying the computation subgraphs.

CHAPTER 3

CGLB: Benchmark Tasks for Continual Graph Learning

This chapter introduces our work on the continual graph learning benchmark (CGLB). Continual learning on graph data, which aims to accommodate new tasks over newly emerged graph data while maintaining the model performance over existing tasks, is attracting increasing attention from the community. Unlike continual learning on Euclidean data (*e.g.*, images, texts, etc.) that has established benchmarks and unified experimental settings, benchmark tasks are rare for Continual Graph Learning (CGL). Moreover, due to the variety of graph data and its complex topological structures, existing works adopt different protocols to configure datasets and experimental settings. This creates a great obstacle to compare different techniques and thus hinders the development of CGL. To this end, we systematically study the task configurations in different application scenarios and develop a comprehensive Continual Graph Learning Benchmark (CGLB) curated from different public datasets. Specifically, CGLB contains both node-level and graph-level continual graph learning tasks under task-incremental (currently widely adopted) and class-incremental (more practical, challenging, yet underexplored) settings, as well as a toolkit for training, evaluating, and visualizing different CGL methods. Within CGLB, we also systematically explain the difference among these task configurations by comparing them to classical continual learning settings. Finally, we comprehensively compare state-of-the-art baselines on CGLB to investigate their effectiveness. Given CGLB and the developed toolkit, the barrier to exploring CGL has been greatly lowered and researchers can focus more on the model development without worrying about tedious work on pre-processing of datasets or encountering unseen pitfalls. The benchmark and the toolkit are available through https://github.com/QueuQ/CGLB.

3.1 Introduction

In real-world applications, many graph data are generated continuously. For instance, in a citation network, new types of papers and their associated citations may emerge and an ideal document classifier needs to continuously adapt its parameters to distinguish the documents of newly emerged research fields [57, 122, 129]; for drug design, new categories or properties of molecules may be encountered and a molecule property prediction model will need to keep updating its parameters to learn new molecule categories or properties [57]. Therefore, given either a continuously expanding graph (with new types of nodes and associated edges) or new categories of graphs continuously emerging, for practical concerns, it is important to develop Continual Graph Learning (CGL) that can accommodate new tasks over newly emerged graph data while maintaining the model performance over existing tasks. However, unlike continual learning on Euclidean data (*e.g.*, images, texts, etc.) that has established benchmarks and unified experimental settings [56, 72, 88, 118, 37, 74, 47, 33], CGL is still immature and with very rare public benchmark tasks being available. Moreover, due to the variety and complexity of graph data, different datasets have been adopted by existing works with different experimental settings. This causes great difficulties in model comparisons and hinders the development of this field. Therefore, it is urgent to create benchmark tasks curated for CGL with standard settings for different graph datasets to minimize the effort spent on tedious data curation as well as the configuration of continual learning tasks, and to avoid encountering potential pitfalls.

In this work, we first provide a systematic taxonomy of different CGL scenarios that include (1) node-level CGL (N-CGL), which deals with node-level prediction on a single growing graph with new types of nodes continuously emerging; and (2) graph-level CGL (G-CGL), which deals with graph-level prediction with new categories of graphs continuously appear. Moreover, we clarify the difference between the currently widely adopted task-incremental learning (task-IL) setting and the underexplored yet more challenging class-incremental learning (class-IL) setting, and point out the proper application scenarios of these two settings. Based on the established taxonomy, we construct the Continual Graph Learning Benchmark

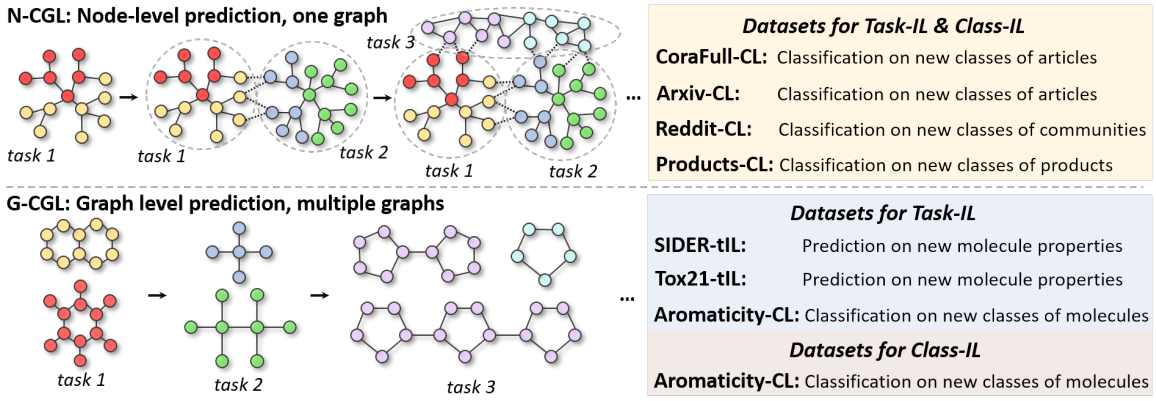


FIGURE 3.1. Overview of 4 different benchmark tasks with N-CGL or G-CGL setting under task-IL or class-IL scenario (applied to different datasets).

(CGLB) with benchmark tasks for each setting. Along with CGLB, we provide a comprehensive toolkit for training, evaluation, and visualization of different CGL models, which could greatly alleviate the burden/risks to investigate CGL problems. Finally, with CGLB and the toolkit, we conduct comprehensive studies on the existing state-of-the-art methods, which reveal the effectiveness of these methods in different scenarios and point out challenges of the existing models under certain settings. With all these efforts, we not only lower the barrier to explore the CGL problem but also provide a systematic overview of the current situation of CGL and point out several challenging directions for future studies.

3.2 Continual Graph Learning Benchmark

In Continual Graph Learning (CGL), a model is expected to learn on a sequence of tasks during the training phase. After learning all tasks, the model is evaluated on each learnt task individually to see how effective the model is on preventing forgetting over existing tasks. Unlike classical continual learning with independent data instances (*e.g.*, images and texts), CGL can be categorized into node-level CGL (N-CGL) and graph-level CGL (G-CGL) settings as shown in Figure 3.1. Tasks in N-CGL are node classification on a single expanding graph with different tasks containing different node classes. In this case, different nodes are connected by edges and thus not independent. Moreover, when new types of nodes emerge, edges connecting different tasks also appear simultaneously, which is a unique problem in

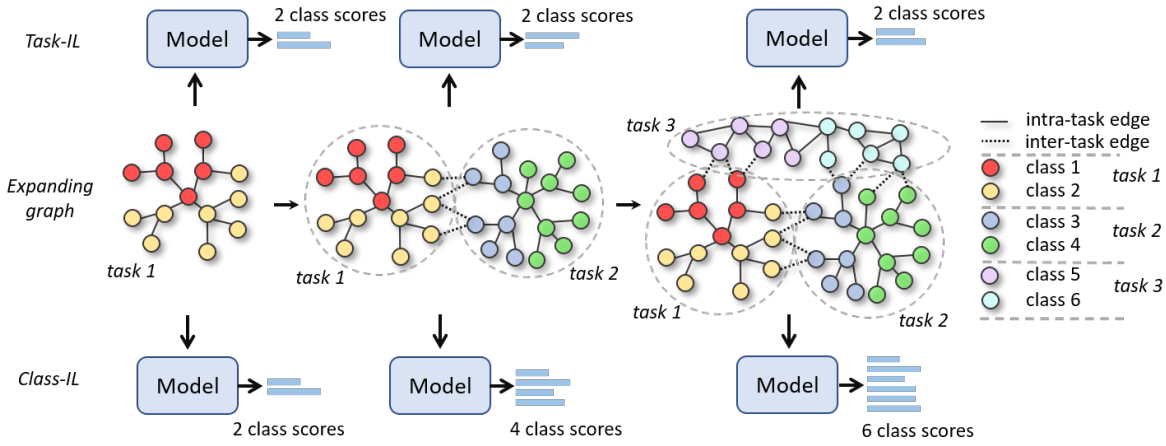


FIGURE 3.2. Illustration of the task-IL and class-IL learning scenario for N-CGL. From left to right, we show the process in which subgraphs (tasks) containing the new classes of nodes are incrementally attached to the existing graph. Different classes are denoted with different colors. The above part shows the task-IL learning scenario, in which the model only needs to distinguish the classes within the current task. The bottom part shows the class-IL scenario, in which the model has to distinguish all the classes from existing tasks.

CGL compared to classical continual learning. The G-CGL, on the other hand, resembles the classical continual learning more, since the examples are individual graphs and without interconnected edges.

3.2.1 Node-level CGL

Node-level CGL (N-CGL) focuses on the node classification problem on an expanding graph as shown in Figure 3.2. Formally, the learning process of N-CGL can be formulated as continually training a model on a sequence of subgraphs (tasks): $\mathcal{S} = \{\mathcal{G}_1, \mathcal{G}_2, ..., \mathcal{G}_T\}$. Each $\mathcal{G}_\tau$ is a newly emerging subgraph of the overall graph with new types of nodes. A $\mathcal{G}_\tau$ is associated with a node set $\mathbb{V}_\tau$ an edge set $\mathbb{E}_\tau$. The edges connecting the new subgraph to the overall graph are denoted as inter-task edges (dashed edges in Figure 3.2). For classification tasks, each node v has a label $\mathbf{y}_v \in \{0, 1\}^C$, where C is the total number of classes.

For classical continual learning, when learning a new task, the model only has access to the data that is related to the current task. However, for N-CGL, attention has to be paid to

the inter-task edges. If the data of existing tasks are strictly inaccessible, then the inter-task edges have to be removed. This is because the message passing mechanism of GNNs would inevitably aggregate information from previous tasks via the inter-task edges. However, a more practical setting is to keep the inter-task edges and only forbid access to the labels of existing tasks. In other words, the supervised learning objective is calculated based on the nodes of the current task, but the node features from the previous tasks are leveraged. In CGLB, we provide both protocols for a comprehensive comparison. We also experimentally explore the difference between these two protocols in Section 3.3.1. In practice, each new subgraph may contain both new types of nodes and old types of nodes. Learning under this scenario is much easier since the data from existing tasks can be leveraged to alleviate the forgetting. Therefore, we do not consider this scenario in this work.

As shown in Table 3.1, we construct four benchmark datasets for N-CGL based on four public data sources: OGB-Arxiv, OGB-Products, Reddit, and CoraFull. We use the suffix -CL (continual learning) to denote that the dataset is for both task-IL and class-IL scenarios. In the following, we also use -tIL or -cIL to denote datasets solely used for task-IL or class-IL setting. To construct these datasets, we first remove the extremely small classes. For example, in OGB-Products, we removed the 47-th class that only has one node and cannot be divided into training, validation, and testing sets. Then, given a dataset with C classes, we split these classes into $\lfloor \frac{C}{K} \rfloor$ groups with a specified order, so that each group containing K classes of nodes. The graph is divided into K subgraphs accordingly. In continual learning, more tasks will result in more severe forgetting problem. This inspires us to set default K as 2 for CGLB, which maximizes the number of tasks to test the limit of given CGL methods. It is also flexible to set K as other integers by simply specifying the parameter when importing the dataset class in CGLB. As for the order of the classes, to facilitate the comparison across different works, the default order is set by CGLB as the original class order within the source dataset. The order can also be set by the users for specific explorations.

TABLE 3.1. The detailed statistics of the constructed benchmark datasets for N-CGL.

Benchmark datasets	CoraFull-CL	Arxiv-CL	Reddit-CL	Products-CL
Data source	CoraFull [65]	OGB-Arxiv [38]	Reddit [35]	OGB-Products [38]
Learning scenario	task-IL & class-IL	task-IL & class-IL	task-IL & class-IL	task-IL & class-IL
# nodes	19,793	169,343	227,853	2,449,028
# edges	130,622	1,166,243	114,615,892	61,859,036
# classes	70	40	40	46
# tasks	30	20	20	23

3.2.2 Graph-level CGL

The majority of graph-level representation learning tasks deal with molecule property prediction. For G-CGL, as shown in Table 3.2, our benchmark datasets are constructed from SIDER, PubChemBioAssayAromaticity, and Tox21. Among these three datasets, different from N-CGL where examples of data (nodes) are connected causing mutual influence and possible inter-task correlation (by inter-task edges) during learning, there is no inter-example influence in G-CGL since the data are individual graphs (*e.g.*, the individual molecule graphs). In the task-IL setting of G-CGL, different tasks correspond to different molecule properties and each task is a binary classification problem. Therefore, the graphs of different tasks in the task-IL setting of G-CGL may overlap, because the same graph can have different properties. Among these datasets, not all graphs are labeled with all properties. Therefore, for each task, the unlabelled graphs are removed. The detailed statistics of these graph datasets are shown in Table 3.2.

3.2.3 Task-incremental and class-incremental learning settings

Based on the fact that whether task indicators are provided to the model during testing, continual learning can be divided into task-incremental (task-IL) and class-incremental (class-IL) settings.

TABLE 3.2. The detailed statistics of the constructed benchmark datasets for G-CGL.

Benchmark datasets	SIDER-tIL	Aromaticity-CL	Tox21-tIL
Data source	SIDER [101]	PubChemBioAssayAromaticity [104]	Tox21 [101]
Learning scenario	task-IL	task-IL & class-IL	task-IL
# graphs	1,427	3,868	7,831
# nodes	48,006	115,061	145,459
# edges	100,912	253,018	302,190
# classes	27	30	12
# tasks	27	15	12

- In task-IL setting, a model requires the task indicator to perform inference on a given task, and the model only needs to distinguish different classes within each task without considering the classes from existing tasks.
- In class-IL setting, task indicators are not provided for the model and the model has to distinguish among all classes from the current and existing tasks.

Concretely, suppose the model has learned on a citation network with a sequence of two tasks {(*physics*, *chemistry*), (*biology*, *math*)}. In class-IL, a given document could come from any of these four classes, and the model is required to classify the given document into one of the four classes. However, in task-IL, a given document comes with a task indicator telling the model whether it is from (*physics*, *chemistry*) or (*biology*, *math*). Then, the model is only required to classify the given document into to (*physics*, *chemistry*) or (*biology*, *math*), without being able to distinguish between *physics* and *biology* or between *chemistry* and *math*. Obviously, for the commonly encountered application scenarios like the citation network or social networks in N-CGL where the tasks are multi-class classification, the class-IL is more practical for N-CGL but is also more challenging. However, for the multi-label classification, like the molecule property prediction tasks in G-CGL, the task-IL and class-IL are equivalent because the model has to give a binary prediction for each task let alone the task orders.

3.2.4 Evaluation metrics and visualization

Different from the traditional learning setting, which typically requires only a single numerical value to indicate the overall performance, CGL is trained and evaluated on multiple tasks and the evaluation protocols should reflect the learning dynamics. Therefore, the most thorough metric to evaluate a CGL model is the performance matrix $\mathrm{M}^p \in \mathbb{R}^{T\times T}$, where $\mathrm{M}^p_{i,j}$ denotes the performance (*e.g.* accuracy, AUC-ROC, etc.) on task j after learning task i.

To better understand the dynamics of the overall performance while learning on the task sequence, we also provide the toolkit to draw the learning dynamics curve, which is a sequence of the average performance (AP): $\left\{\frac{\sum_{j=1}^{i}\mathrm{M}^p_{i,j}}{i}|i=1,...,T\right\}$ and a sequence of the average forgetting (AF): $\left\{\frac{\sum_{j=1}^{i-1}\mathrm{M}^p_{i,j}-\mathrm{M}^p_{j,j}}{i-1}|i=2,...,T\right\}$ when the number of learned tasks varies. To use a single numerical value to quantify the overall learning performance, the AP and AF after learning all T tasks can be used. In our library, the output is formatted in standard forms and can be directly evaluated with these metrics with the evaluation protocol with a one-line function.

3.3 Baseline Implementations and Result Analysis

In this section, we tested state-of-the-art continual learning (CL) methods on CGLB. These CL methods are implemented based on GNN backbones. A brief introduction of the implemented CL methods is as below:

(1) **Bare model** denotes the backbone GNN without continual learning technique. Therefore, this can be viewed as the lower bound on the continual learning performance.

(2) **Elastic Weight Consolidation (EWC) [44]** adds a quadratic penalty on the model weights according to their importance to the previous tasks to maintain its performance on existing tasks.

(3) **Memory Aware Synapses (MAS) [3]** is also based on regularization. Different from EWC, MAS evaluates the parameter importance according to the sensitivity of the predictions on the parameters.

(4) **Gradient Episodic Memory (GEM) [59]** stores representative data in the episodic memory. During learning, GEM modifies the gradients of the current task with the gradient calculated with the stored data to prevents the loss of the previous tasks from increasing.

(5) **Topology-aware Weight Preserving (TWP) [57]** adds a penalty to preserve the topological information of the previous graphs.

(6) **Learning without Forgetting (LwF) [52]** minimizes the discrepancy between the logits of the old model and the new model (knowledge distillation) to preserve knowledge from the old tasks.

(7) **Experience Replay GNN (ER-GNN) [129]** integrates memory-replay to GNNs by storing representative nodes selected from previous tasks.

(8) **Joint Training** does not follow the continual learning setting and trains the backbone GNN on all tasks simultaneously. Therefore, Joint Training has no forgetting problems and its performance can be viewed as the upper bound for continual learning.

3.3.1 N-CGL under the task-IL scenario

In this subsection, we tested state-of-the-art methods on CGLB under the widely adopted task-IL scenario. We also studied the influence of the inter-task edges on the performance, which is currently ignored by the existing works. First, in Table 3.3, we show the final AP and AF (defined in Section 3.2.4) of each method after learning each entire task sequence. On average, the Bare model without any continual learning technique performs the worst, and Joint training performs the best. The AF is inapplicable to joint trained models because they do not follow the continual learning setting and are simultaneously trained on all tasks. In terms of the final AF, all methods significantly outperform the Bare model especially the regularization based ones like EWC, MAS, and TWP, which demonstrates that the regularization methods indeed can alleviate the forgetting problem. But in terms of the final AP, the methods with high AF (little forgetting problem) may not necessarily have high AP. In some circumstances, a higher AF may be accompanied by a lower AP. For example, on Reddit-CL, MAS obtains

TABLE 3.3. Performance comparisons under task-IL without inter-task edges on different datasets (↑ higher means better).

C.L.T.	CoraFull-CL		Arxiv-CL		Reddit-CL		Products-CL	
	AP/% ↑	AF/% ↑	AP/% ↑	AF /% ↑	AP/% ↑	AF /% ↑	AP/% ↑	AF /% ↑
Bare model	56.8±0.9	-39.4±0.8	64.8±2.5	-25.0±1.5	79.5±13.0	-11.7±2.9	64.4±2.3	-31.1±2.6
EWC [44]	75.3±7.1	-19.0±7.0	65.4±2.3	-20.1±2.1	83.9±11.9	-2.0±1.0	87.0±0.9	-1.7±0.9
MAS [3]	92.7±0.2	-0.5±0.1	88.5±0.5	-0.0±0.4	61.1±4.3	-0.5±0.7	80.6±2.4	-13.7±2.4
GEM [59]	91.5±0.4	-1.9±0.5	79.9±0.3	-5.2±0.2	98.9±0.0	-0.5±0.1	87.7±1.1	-7.0±1.2
TWP [57]	92.4±0.2	-0.9±0.3	86.7±0.1	-2.1±0.3	78.0±13.4	-0.2±0.3	81.8±2.2	-0.3±0.5
LwF [52]	57.8±0.4	-38.3±0.5	65.7±2.9	-24.2±3.6	98.6±0.1	-0.0±0.3	86.3±0.1	-0.5±0.1
ER-GNN [129]	69.9±1.3	-25.5±1.4	86.4±0.2	0.5±0.3	97.5±1.5	2.6±3.7	86.4±2.3	11.7±1.2
Joint	96.0±0.1	-	90.3±0.2	-	99.5±0.3	-	95.3±0.4	-

TABLE 3.4. Performance comparisons under task-IL on different datasets with inter-task edges (↑ higher means better).

C.L.T.	CoraFull-CL		Arxiv-CL		Reddit-CL		Products-CL	
	AP/% ↑	AF/% ↑	AP/% ↑	AF /% ↑	AP/% ↑	AF /% ↑	AP/% ↑	AF /% ↑
Bare model	58.0±2.1	-38.5±1.9	57.7±1.2	-31.2±1.0	80.0±12.5	-11.5±3.7	60.5±3.6	-34.6±2.7
EWC [44]	72.6±1.3	-23.1±0.9	70.1±0.8	-15.7±1.2	83.5±0.8	-2.3±0.9	85.6±1.1	-3.4±0.4
MAS [3]	92.4±0.3	-1.8±0.5	88.0±1.0	-0.2±0.7	61.7±5.5	-0.2±1.9	78.4±1.6	-15.7±2.3
GEM [59]	92.8±0.2	-2.0±0.6	78.7±1.2	-4.5±0.6	98.2±0.4	-0.6±0.5	87.9±0.6	-6.9±0.4
TWP [57]	89.9±0.5	-3.9±0.3	88.4±0.9	-0.5±1.0	78.3±0.8	-0.3±1.3	81.5±0.5	-0.6±0.3
LwF [52]	63.7±3.2	-32.5±4.2	65.3±2.1	-26.0±1.7	98.3±0.3	-1.1±0.2	86.7±1.0	-0.2±0.3
ER-GNN [129]	71.2±1.3	-23.2±2.0	82.3±1.9	-1.2±1.2	97.0±0.4	2.5±0.4	86.8±1.1	10.4±0.8
Joint	93.5±0.3	-	97.0±0.2	-	99.6±0.3	-	92.8±0.5	-

significantly higher AF, but lower AP compared to the Bare model. This reveals the downside of the regularization term, which is the lower adaption ability to new tasks. In contrast, although the Bare model experience severe forgetting problem, its adaptation to new tasks is free without any constraint. This difference is also further analyzed in Section 3.3.2 with the learning dynamics curve and the performance matrices.

In Table 3.4, we show the results of N-CGL under task-IL scenario with inter-task edges being kept. Comparing Table 3.3 and 3.4, the inter-task edges are indeed causing significant difference in the performance. However, according to the results, the difference cannot be simply summarized as one is more superior than the other, because the influence factors are multi-fold and the performance difference also varies across different datasets and methods. Specifically, multiple factors including both beneficial and harmful ones are governing the performance difference, which is analyzed below.

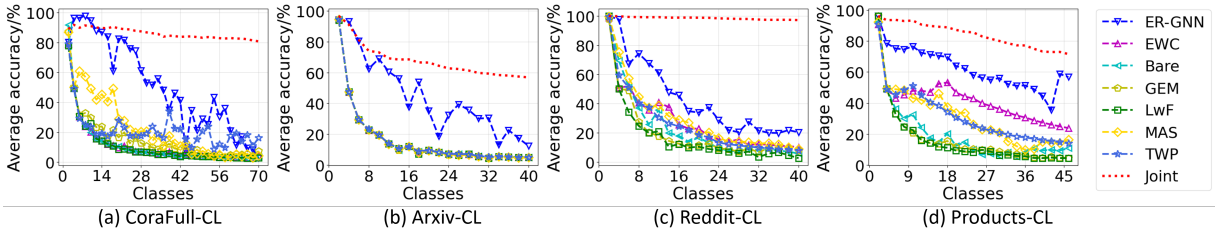


FIGURE 3.3. Dynamics of the AP during learning on the task sequences of different datasets.

(1) Each node is predicted based on its neighborhood by GNNs. The inter-task edges alter the neighborhoods of the boundary nodes (forming edges to new subgraphs) of the previous tasks, causing a concept drift. Since the training on these affected nodes is not repeated, this could harm the performance.

(2) Due to the message passing mechanism of GNNs, the inter-task edges also break the restriction on the access to the information of the previous tasks. Real-world graphs often exhibit the 'small-world' property which means the shortest path between any two nodes is mostly short. Therefore, the inter-task edges are likely to propagate a significant amount of information from old tasks to new tasks. This effect may alleviate the forgetting problem and be beneficial for performance.

(3) With inter-task edges, more information about the neighborhood patterns is provided for the boundary nodes. By examining Joint training, which is not influenced by the two effects mentioned above, we can find that the influence of inter-task edges highly depends on the datasets.

In a nutshell, the inter-task edges are introducing contradictory factors influencing the performance, and whether the net influence is beneficial or harmful highly depends on the properties of the datasets.

Overall, existing methods could perform well in N-CGL under the task-IL scenario, and some are even close to Joint training. However, for N-CGL, it is more practical (yet challenging) to conduct a class-IL setting, as explained in Section 3.2.1, which is therefore studied in the next subsection.

TABLE 3.5. Performance comparisons under class-IL on different datasets without inter-task edges (↑ higher means better).

C.L.T.	CoraFull-CL		Arxiv-CL		Reddit-CL		Products-CL	
	AP/% ↑	AF/% ↑	AP/% ↑	AF /% ↑	AP/% ↑	AF /% ↑	AP/% ↑	AF /% ↑
Bare model	2.9±0.1	-94.8±0.4	4.9±0.0	-88.4±0.3	5.9±0.9	-97.9±1.8	7.6±0.6	-88.7±0.5
EWC [44]	4.7±0.3	-93.0±0.2	4.9±0.2	-89.2±0.4	10.3±6.3	-33.2±14.6	23.8±2.2	-21.7±3.9
MAS [3]	4.5±0.1	-91.0±0.3	4.9±0.1	-87.0±0.2	9.2±7.6	-23.1±14.6	16.7±2.6	-57.0±24.8
GEM [59]	7.6±0.3	-89.5±0.4	4.8±0.5	-87.8±0.2	5.0±0.2	-99.4±0.2	4.5±0.8	-94.7±0.2
TWP [57]	16.2±0.5	-78.9±0.9	4.9±0.1	-89.4±0.3	8.0±2.9	-18.8±5.1	14.1±2.1	-11.4±1.3
LwF [52]	2.9±0.1	-94.7±1.0	4.9±0.1	-87.5±0.2	14.4±0.4	-88.7±0.3	4.5±0.3	-95.7±0.5
ER-GNN [129]	2.9±0.1	-94.6±0.2	12.3±3.1	-79.9±3.3	20.4±2.6	-82.7±2.9	56.7±0.3	-33.3±0.5
Joint	80.8±0.1	-	56.8±0.3	-	97.1±0.1	-	71.5±0.1	-

3.3.2 N-CGL under the class-IL scenario

Different from the task-IL, class-IL is currently rarely studied for CGL. With the final AP and AF showed in Table 3.5, the first impression is that the performance is much worse compared to the task-IL scenario. This is not solely caused by the forgetting problem but also by the increased number of classes (difficulty) in each task. For example, on CoraFull-CL with 35 tasks and 2 classes per task, the task-IL scenario only requires the model to perform a 2-class classification for each task. While in class-IL, the final evaluation requires the model to perform a 70-class classification. Despite the difficulty in the increasing number of classes, forgetting is still the governing factor in the performance, since Joint training obtains pretty high performance. To further understand the learning dynamics with the task sequence, we show the learning dynamics curve and visualize the performance matrix as defined in Section 3.2.4. The learning curves, as shown in Figure 3.3, reveal the decreasing patterns of the AP for different methods. The long task sequence constructed in CGLB with tens of tasks is indeed challenging. Many methods with final AP close to the Bare model in Table 3.5 outperform the Bare model when fewer tasks are learned. The performance is also highly dependent on the datasets. On the most challenging Arxiv-CL, most methods experience strong forgetting problems, and even the Joint training obtains a much lower performance compared to other datasets. Since the memory-replay based method (ER-GNN) performs better than the regularization based techniques, this implies that the distribution discrepancies across different tasks of Arxiv-CL are strong and the regularization cannot simultaneously maintain the performance on previous tasks and keep the flexibility to adapt to new tasks.

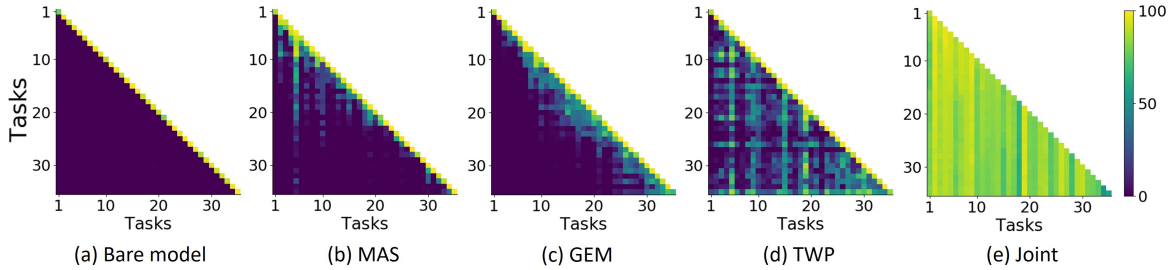


FIGURE 3.4. Visualization of the performance matrices of different continual learning methods on CoraFull-CL. Each entry in these matrices represent the performance on task j (column) after learning task i (row).

In contrast, different methods on Products-CL exhibit reasonable improvement compared to the Bare model. Overall, under the class-IL scenario, memory replay based techniques (ER-GNN) appear to be more effective compared to other CGL techniques.

To gain a better understanding of the behaviors of different baselines, we visualize the performance matrices, the most thorough evaluation metric, of several representative baselines in Figure 3.4. We first observe two representative patterns of the Bare model and the Joint training. The Bare model has no problem learning each new task (the diagonal entries, better viewed with zooming in), but experiences a catastrophic forgetting of the previously learned tasks. In contrast, Joint training demonstrates a perfect performance that both learn well on new tasks and exhibit no forgetting on each existing task (each column denotes the performance change of one task along with the learning on the task sequence). The baselines, *i.e.*, MAS, GEM, and TWP, lie somewhere between these two extreme cases. Since they all have certain constraints on the changes of the model parameters, the ability to adapt to new tasks is weaker than the Bare model (*i.e.* the diagonal entries have smaller values). But the regularization also enables the model to maintain its performance over existing tasks to a certain extent. By observing each column, we could see that the performance of each task gradually decreases along the learning process, while the Bare model experiences an abrupt decrease on the latest existing task immediately after learning the new task. Due to space limitations, the results of other techniques are provided in Section 3.5.2.

TABLE 3.6. Performance comparisons on different graph-level prediction datasets (↑ higher means better).

C.L.T.	SIDER-tIL		Tox21-tIL		Aromaticity-CL		Aromaticity-CL	
	task-IL		task-IL		task-IL		class-IL	
	AP ↑	AF ↑	AP ↑	AF ↑	AP/% ↑	AF /% ↑	AP/% ↑	AF /% ↑
Bare model	0.577±0.021	-0.070±0.023	0.717±0.012	-0.138±0.015	53.2±3.8	-42.4±7.5	5.4±0.2	-66.3±2.0
EWC [44]	0.604±0.001	-0.012±0.009	0.811±0.005	-0.015±0.002	58.0±5.0	-22.8±2.5	3.1±1.5	-25.9±0.7
MAS [3]	0.627±0.018	0.001±0.001	0.798±0.009	-0.022±0.004	63.8±1.3	-24.0±2.3	7.8±1.0	-80.5±2.1
GEM [59]	0.655±0.010	0.037±0.002	0.825±0.008	0.024±0.012	70.8±0.7	-3.1±0.5	9.3±2.4	-75.6±1.1
TWP [57]	0.611±0.015	-0.005±0.002	0.720±0.024	-0.037±0.018	54.5±1.3	-21.4±0.2	6.5±1.6	-45.4±7.7
LwF [52]	0.624±0.003	-0.008±0.009	0.673±0.009	-0.175±0.002	56.7±2.9	-23.9±0.6	4.6±0.6	-65.4±4.5
Joint	0.692±0.068	-	0.826±0.005	-	75.4±6.0	-	77.2±1.3	-

3.3.3 G-CGL

In this subsection, we study the G-CGL for both task-IL and class-IL scenarios. The performance matrix contains the AUC-ROC of each task for the task-IL setting, and the accuracy of each task for the class-IL setting. As shown in Table 3.6, the task-IL is still relatively easier and all baseline methods gain reasonable improvement against the bare model. In contrast, all baseline methods perform badly under the class-IL scenario. Since the Joint training obtains satisfying performance, it is clear that the failures of the baselines are not caused by the backbone GNNs but are caused by the forgetting problem. Different tasks of G-CGL correspond to different topology distributions of graphs. The topology distributions appear to be highly different, which is also verified by the results that GEM often encounters the no solution error. Essentially, GEM is developed based on clipping the gradients of the current task so that the update directions do not lead to the loss increase of the previous tasks. Therefore, the no solution error implies that the task distributions are so different that learning some of the new tasks will inevitably increase the loss of some previous tasks. This error is not encountered in N-CGL, which implies that the distribution discrepancy is more severe in G-CGL. More detailed studies to further justify the above analysis could be obtained with the learning dynamics curves and the performance matrices, which are included in Section 3.5.2.

3.4 Additional Details on the Experimental Settings and Benchmark Construction

In this section, we give further details on the experimental settings and details on the construction of CGLB.

3.4.1 Additional Details on the Experimental Settings

Additional experimental setting details include the indices of the deleted classes for each dataset, the used devices, the train-validation-test splitting for each task, and details on training and validation. For the N-CGL datasets, we remove the 41-th class of Reddit-CL, and the 47-th class of Products-CL. This aims to ensure an even number of classes for each dataset to be divided into a sequence of 2-class tasks. Moreover, the 47-th class of Products-CL contains only one node, and cannot be split for training, validation, and test. For the G-CGL datasets, classes removed from Aromaticity-CL are {2, 3, 4, 8, 35, 36, 37, 38, 39, 40, 41} since they contain less than 20 examples and are causing difficulties for model training. The other 30 classes of Aromaticity-CL are kept and constructed as 15 tasks. No classes are removed from SIDER-tIL and Tox21-tIL. The statistics in Tables 3.1 and 3.2 are obtained after the class removal. For the N-CGL datasets, the train-validation-test splitting ratios are 60%, 20%, and 20%, and transductive setting is adopted for each task. The train-validation-test splitting is obtained by random sampling, therefore the performance may be slightly different with splittings from different rounds of random sampling. We provide the set of splitting used in our experiments on our GitHub page as a reference. For the G-CGL datasets, the ratios are 80%, 10%, and 10%. In our code, we provide a framework for conveniently conduct grid-search over candidate hyper-parameters and find the best performing combination on the validation set. Then the selected model is also automatically evaluated on the testing set. Details on the usage can be found in our GitHub page. In our experiments, the hyper-parameter candidates we used are summarized in Table 3.7. The name of the hyper-parameters are consistent with the names in our code. Details of the parameters could be found in the original papers. Among the hyper-parameters of all the methods, the memory budget of the

memory based methods (GEM and ERGNN) should be cared. Generally, larger memory budget leads to better performance. However, larger budget also consumes more space, which may not be practical in real-world applications. Therefore, unlike the other hyper-parameters that can be determined by validation, the memory budget is also determined by the available space in practical scenarios. Besides, the dataset splittings and the method hyper-parameters, the batch size and the number of training epochs are also factors influencing the continual learning performance. Smaller batch sizes or larger number of training epochs lead to more iterations to train the model, and the model is more adapted to a given task (more forgetting on previous tasks). Therefore, keeping a consistent batch size is an important factor when making comparisons across different models. In NCGL experiments, we use full batch for the small NCGL datasets (CoraFull-CL and Arxiv-CL) and the GCGL datasets, and use 2,000 as the batch size for learning on large NCGL datasets (Reddit-CL and Products-CL). For GCGL experiments, we set the batch size as 128, and the number of training epochs as 100. For the two multi-label classification datasets (SIDER-tIL and Tox21-tIL), early stopping is applied to ensure a stable performance. All experiments are repeated 5 times on one Nvidia Titan Xp GPU.

TABLE 3.7. Hyper-parameter candidates used for grid search.

EWC [44]	memory_strength: [1, 100, 10000, 1000000]
MAS [3]	memory_strength: [1, 100, 10000, 1000000]
GEM [59]	memory_strength: [0.05,0.5,5]; n_memories: [10, 100,1000]
TWP [57]	lambda_l: [100,10000] ; lambda_t: [100,10000]; beta: [0.01,0.1]
LwF [52]	lambda_dist: [0.1,1.,10.] ; T: [0.2,2,20]
ER-GNN [129]	budget: [10,100]; d: [0.05, 0.5, 5.0]; sampler: [CM]

3.4.2 Additional Explanations on the Evaluation Metrics

In this subsection, we explain the evaluation metrics in details. The explanations will cover every step in the computation of AP and AF from the original performance matrix, so that the rationale behind the AP, AF and the performance matrix will be clarified.

The rationale behind AP and AF is to measure a model's average performance/forgetting on all learnt tasks after learning a sequence of tasks. In the following, we will separately explain AP and AF with details and examples.

In a performance matrix M^p, an entry $M^p_{i,j}$ denotes the model performance on a learnt task j ($j \leqslant i$) after learning the i-th task (i.e., the model has been trained over the sequence of tasks from 1 to i). Accordingly, the i-th row of the matrix M^p, i.e. $M^p_{i,j}, j = 1, ..., i$ reflects the performance on each learnt task j ($j = 1, ..., i$) after learning the i-th task.

Therefore, the average performance (AP) after learning the i-th task is defined as the average of the i-th row of the matrix ($M^p_{i,j}, j = 1, ..., i$), because it reflects the model's average performance over all learnt tasks after learning a sequence of tasks (i.e., from the 1-st task to the i-th task). As we could see, an AP can be computed after learning each new task, and the sequence of APs denotes how the overall performance varies with new tasks constantly coming in (i.e., the learning dynamics curves shown in Figure 3.3).

When a single numerical value is required to denote the model's overall performance, the AP computed after learning the entire sequence of tasks (i.e., $\frac{\sum_{j=1}^{T} M^p_{T,j}}{T}$ computed over the last row of the performance matrix, $M^p_{T,j}, j = 1, ..., T$) is used, which is the average model performance over all learnt tasks after the model has been trained over the sequence from the 1-st task to the final task. A high AP denotes that the model's overall performance on previous tasks is good after learning all the tasks sequentially (i.e., little forgetting issue). While a lower AP denotes that the model has more severe forgetting problem.

The average forgetting (AF) is supported by the same rationale. Specifically, a diagonal entry $M^p_{j,j}$ denotes the performance of a model when it has just learnt the task j (i.e., before the performance on task j is degraded because of the forgetting issue). The model will then keep learning the following tasks $j + 1$, $j + 2$, ..., etc.. After learning each following new task, the model is tested again on task j, and the performance on task j becomes $M^p_{j+1,j}$, $M^p_{j+2,j}$, ..., etc.. At a specific step i, when the model has just learnt task i ($i > j$), the performance on task j becomes $M^p_{i,j}$. Due to the forgetting issue (learning new tasks may interfere with the performance on task j), $M^p_{i,j}$ may be lower than $M^p_{j,j}$, and $M^p_{i,j} - M^p_{j,j}$ can quantitatively measure the forgetting. Accordingly, negative $M^p_{i,j} - M^p_{j,j}$ denotes that the performance on task j is negatively affected (forgetting on task j) by the following tasks from $j + 1$ to i, and larger $|M^p_{j+1,j} - M^p_{j,j}|$ denotes more severe forgetting. It is also possible that $M^p_{j+1,j} - M^p_{j,j}$ is positive, denoting that the learning on the following tasks has a positive influence on task j,

which is rare according to the experimental results. Similar to AP, after learning each new task, we could compute an AF over all learnt tasks, and the sequence of AFs reflects the learning dynamics from the forgetting perspective. To use a single numerical value to denote the average forgetting over all learnt tasks after learning the final task T, we could use the AF computed after learning the T-th task, i.e. $\frac{\sum_{j=1}^{T-1} \mathrm{M}_{T,j}^{p} - \mathrm{M}_{j,j}^{p}}{T-1}$.

3.4.3 Class Imbalance Problem

The class imbalance problem is often severe in graph data, and the model may collapse and give trivial predictions biased to the dominating classes. Moreover, due to the topological connections among the data and the severeness of the imbalance problem, it may not be practical to balance the data by choosing equal number of nodes from each class. For example, the largest class in Products-CL has 668,950 nodes, and the smallest class has 1 node. In this situation, selecting a equal number of nodes from each class will either make it necessary to delete many classes without enough number of nodes or selecting a very small amount of nodes from each class to ensure that all classes have a same size. Besides, deleting graph nodes also change the original topology and the relations among the remaining nodes, and is better avoided.

Therefore, during training, we would re-scale the loss of the nodes according to the sizes of their corresponding classes. Suppose the set of all classes of a training set is $\mathcal{C}$, and the size of each class is $\{n_c \mid c \in \mathcal{C}\}$. Then, the loss calculated on a node set $\mathbb{V}$ without balancing can be formulated as,

$$\mathcal{L} = \sum_{v \in \mathbb{V}} l(f(v), y_v), \tag{3.1}$$

where $f(v)$ denotes the prediction of v, and y_v denotes the label of v. The balanced loss is then formulated as,

$$\mathcal{L} = \sum_{v \in \mathbb{V}} l(f(v), y_v) \cdot s_{y_v}, \tag{3.2}$$

where $s_{y_v} = \frac{n_c}{\sum_{i \in \mathcal{C}} n_i}$ when $y_v = c$.

In testing, the evaluation of the model also considers the class imbalance problem. We will first calculate the performance within each class, and then average the results over all classes. In this way, the classes of different sizes contribute equally to the performance.

3.4.4 Classifier for Class-IL Scenario

In a standard classification task, the number of the output heads (number of output logits) of a classifier equals the number of possible classes in the dataset, and is fixed before training. However, in class-IL scenario of CGL, since new classes come in constantly, the number of output heads cannot be set beforehand but has to continually increase.

3.4.5 Task Statistics

Besides the dataset statistics provided earlier, in this subsection, we provide detailed statistics for each task, i.e., the numbers of nodes/graphs/edges in each class for all GCGL tasks, and the numbers of nodes/edges in each class for all NCGL tasks. The statistics are shown Table 3.8,3.9,3.10,3.11,3.12,3.13,3.14.

3.5 Additional Experimental Results

In this section, we provide additional results to analyze the performance of different baseline methods on CGLB. The results will be mainly based on the visualization of the performance matrices, which is the most thorough metric to evaluate a continual learning model, of different methods on different benchmark tasks.

3.5.1 Investigation on the Influence of GNN backbones

In this section, we investigate the influence of the GNN backbones on the performance of continual graph learning. To ensure a fair comparison, all GNNs are configured as 2-layer. From Table 3.15, our first finding is although adopting different backbones result in different

class	0	1	2	3	4	5	6	7	8	9
# nodes	257	52	243	378	63	305	404	663	240	342
# edges	3066	976	3540	4404	966	2350	4968	12584	4488	2884
class	10	11	12	13	14	15	16	17	18	19
# nodes	141	223	102	521	341	138	115	111	80	435
# edges	2226	2892	1108	9078	3548	1468	1430	1762	416	3842
class	20	21	22	23	24	25	26	27	28	29
# nodes	420	254	414	196	334	315	284	783	113	466
# edges	6722	2040	4492	2328	4266	4266	2510	18676	1544	5606
class	30	31	32	33	34	35	36	37	38	39
# nodes	221	376	154	855	576	84	293	163	125	564
# edges	3300	4014	1708	8116	8554	592	5694	3530	1034	5490
class	40	41	42	43	44	45	46	47	48	49
# nodes	280	205	94	53	129	370	122	74	557	285
# edges	1982	2106	1240	472	1572	3716	1272	840	7384	3054
class	50	51	52	53	54	55	56	57	58	59
# nodes	72	625	501	650	99	473	324	928	212	301
# edges	1138	10772	5176	6834	750	5974	3218	8566	2488	6454
class	60	61	62	63	64	65	66	67	68	69
# nodes	116	220	165	291	147	91	137	84	15	29
# edges	2000	3434	1798	3882	1534	1068	1314	746	82	340

TABLE 3.8. Number of nodes and edges in each class of CoraFull-CL.

class	0	1	2	3	4	5	6	7	8	9
# nodes	565	687	4839	2080	5862	4958	1618	589	6232	2820
# edges	4001	4697	26310	12439	46538	34558	7466	2447	41732	22296
class	10	11	12	13	14	15	16	17	18	19
# nodes	7869	750	29	2358	597	403	27321	515	749	2877
# edges	77876	5180	136	37716	4670	2668	733536	5005	4925	12172
class	20	21	22	23	24	25	26	27	28	29
# nodes	2076	393	1903	2834	22187	1257	4605	4801	21406	416
# edges	11062	1525	10036	15459	421129	12827	39656	43865	262381	2345
class	30	31	32	33	34	35	36	37	38	39
# nodes	11814	2828	411	1271	7867	127	3524	2369	1507	2029
# edges	250277	23863	2681	6243	75707	432	28697	15291	11345	11297

TABLE 3.9. Number of nodes and edges in each class of Arxiv-CL.

continual learning performance, the relative performance comparison (ranking) of different continual learning techniques are similar across different backbones except for EWC and MAS with GAT backbone. The performance of EWC and MAS with GAT backbone exhibit a contrary pattern compared to the performance obtained with the other two backbones. Specifically, EWC exhibits more severe forgetting issues with GCN and GIN compared to GAT. On the contrary, MAS exhibits more severe forgetting issues with GAT compared to

class	0	1	2	3	4	5	6	7	8	9
# nodes	13101	3550	3302	15181	2322	3597	3952	2138	11187	2246
# edges	9333976	2207936	2170698	4435498	1284924	8838256	2876904	1981664	17567126	1577840
class	10	11	12	13	14	15	16	17	18	19
# nodes	4928	2964	1696	2731	4854	28272	1003	2639	13999	10308
# edges	4377476	3370058	3875108	1871030	9203184	23998718	1035980	3237128	16823834	7086846
class	20	21	22	23	24	25	26	27	28	29
# nodes	1596	4066	8222	12146	328	1659	4239	5962	4673	5101
# edges	1750498	5087278	2125064	4270774	148988	2005018	3190862	5055124	1505350	3147048
class	30	31	32	33	34	35	36	37	38	39
# nodes	2846	4570	1575	4960	3429	4202	4180	4233	12797	3099
# edges	2293504	8151694	836924	13468276	8032124	6091244	5871398	5330018	19629974	1190018
class	40									
# nodes	5112									
# edges	2896422									

TABLE 3.10. Number of nodes and edges in each class of Reddit-CL.

class	0	1	2	3	4	5	6	7	8	9
# nodes	114294	109832	116043	151061	668950	40715	158771	172199	110796	67358
# edges	12896708	14838746	11449058	15576466	48891834	5984624	26375396	14340184	7605922	8296044
class	10	11	12	13	14	15	16	17	18	19
# nodes	52345	32937	131886	101541	3079	26911	83594	42337	49019	17438
# edges	7608804	1533830	12407580	9797510	151210	3758376	9552274	6297586	5733784	2187382
class	20	21	22	23	24	25	26	27	28	29
# nodes	22575	80795	879	3653	45406	3024	553	259	1969	1561
# edges	2247434	8065332	74388	877932	3593542	240336	78188	14760	106084	81154
class	30	31	32	33	34	35	36	37	38	39
# nodes	277	418	513	29	154	44	630	514	91	37
# edges	11672	30766	35648	3108	8066	2336	62204	44886	5972	3288
class	40	41	42	43	44	45	46			
# nodes	6	61	32500	1399	566	9	1			
# edges	876	17578	6229472	194368	122826	818	208			

TABLE 3.11. Number of nodes and edges in each class of Products-CL.

class	0	1	2	3	4	5	6	7	8	9
# nodes	23472	34872	873	27456	40354	35359	44618	8211	37300	24126
# graphs	743	996	22	876	1151	997	1298	251	1024	727
class	10	11	12	13	14	15	16	17	18	19
# nodes	14057	44142	10667	6454	38814	27669	45146	7740	35994	36826
# graphs	376	1292	323	213	1108	885	1318	253	1006	1060
class	20	21	22	23	24	25	26			
# nodes	34027	31894	4694	21166	35159	45489	34019			
# graphs	1016	911	125	659	988	1304	946			

TABLE 3.12. Number of nodes and edges in each class of SIDER-tIL.

class	0	1	2	3	4	5	6	7	8	9
# nodes	7820	5808	14428	7378	15835	7596	4002	18679	5340	7096
# graphs	309	237	768	300	793	350	186	942	264	372
class	10	11								
# nodes	20099	10151								
# graphs	918	423								

TABLE 3.13. Number of nodes and edges in each class of Tox21-tIL.

class	0	1	2	3	4	5	6	7	8	9
# nodes	3139	0	0	0	1553	3007	3405	185	1670	3144
# graphs	148	0	0	0	60	149	150	13	83	148
class	10	11	12	13	14	15	16	17	18	19
# nodes	3364	3553	3914	3449	3686	3967	4066	4194	4629	4155
# graphs	149	150	150	144	150	150	149	149	150	149
class	20	21	22	23	24	25	26	27	28	29
# nodes	4355	4544	4824	4927	5202	4823	4820	4946	5189	3954
# graphs	149	148	149	146	145	144	136	136	132	94
class	30	31	32	33	34	35	36	37	38	39
# nodes	4817	3425	2749	1591	824	2730	1132	841	543	203
# graphs	106	74	52	29	17	15	16	6	6	1

TABLE 3.14. Number of nodes and edges in each class of Aromaticity-CL.

TABLE 3.15. Performance comparisons with different backbone GNNs under task-IL without inter-task edges (↑ higher means better).

C.L.T.	GCN		GAT		GIN	
	AP/% ↑	AF/% ↑	AP/% ↑	AF /% ↑	AP/% ↑	AF /% ↑
Bare model	61.7±3.8	-28.2±3.3	63.7±1.6	-27.3±1.7	65.1±1.6	-23.9±1.5
EWC [44]	78.8±2.7	-5.0±3.1	75.1±0.5	0.1±0.2	63.2±3.0	-21.5±3.2
MAS [3]	88.4±0.2	-0.0±0.0	80.5±1.5	-8.6±1.2	87.9±0.4	-0.1±0.1
GEM [59]	87.3±0.6	2.8±0.3	79.2±0.4	-5.7±0.3	80.6±0.8	-3.6±1.1
TWP [57]	86.0±0.8	-2.8±0.8	86.6±0.2	-1.5±0.3	88.3±0.6	-0.6±0.8
LwF [52]	84.2±0.5	-3.7±0.6	54.5±0.7	-36.9±0.8	84.0±1.7	-3.7±1.3
ER-GNN [129]	86.7±0.1	11.4±0.9	88.4±0.4	4.2±0.8	89.3±0.1	5.0±0.4
Joint	90.3±0.2	-	88.8±0.4	-	88.9±0.1	-

GCN and GIN. This phenomenon is reasonable considering that the structure of GCN and GIN are very similar, while GAT follows a different framework. Both EWC and MAS are regularization based methods with different strategies for evaluating the importance of the model weights. This implies that for different backbones, the strategies for regularizing the model weights matter a lot and should be carefully designed. As for the bare model, bare GCN and GIN models also have similar performance, while bare GAT models perform worse.

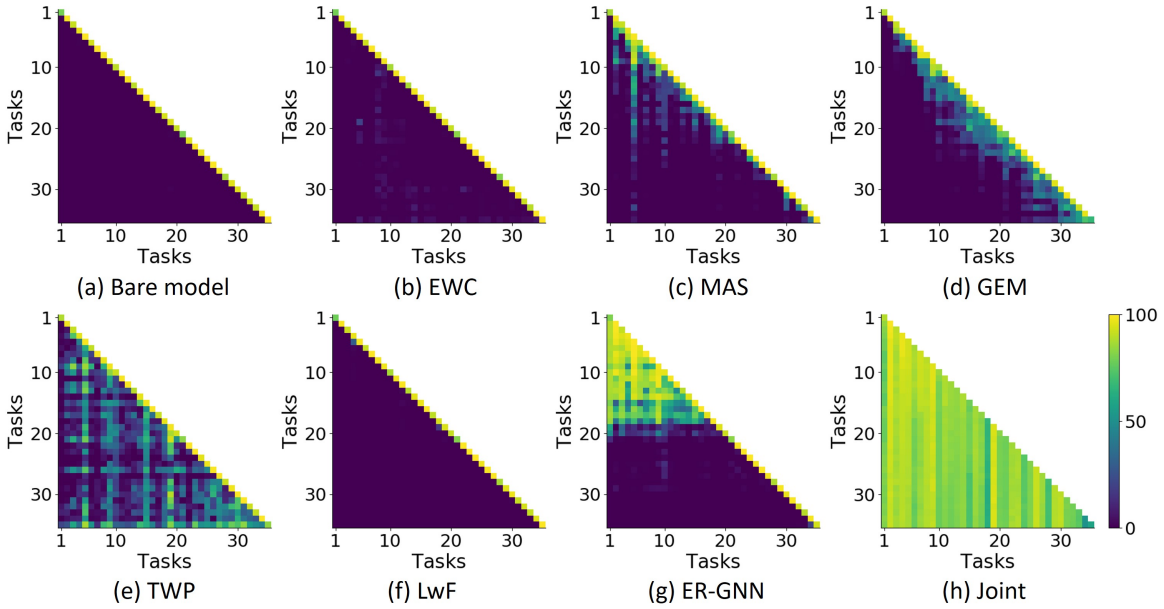


FIGURE 3.5. Visualization of the performance matrices of different methods on CoraFull-CL under class-IL setting.

This phenomenon is closely related to the complexity of different models. The trainable part of GCN and GIN models are mainly fully connected layers, while GAT has a much more complex structure with the trainable attention mechanism. More complex models tend to overfit the new tasks with more severe forgetting issues on previous tasks. Finally, the jointly trained GCN, GAT, and GIN exhibit similar performance with tiny variations.

3.5.2 Additional Results on the Performance Matrix Visualization

We have visualized the performance matrices of several representative methods to show the details of the forgetting behavior under the class-IL scenario. In this subsection, we visualize the performance matrices of all different methods.

In Figure 3.5, the performance of the bare model is typical well for new tasks which are just learned, and bad for the old tasks since they are totally forgotten. The results of EWC exhibit certain improvement over the bare model, although not significant. MAS, which follows a similar regularization strategy, exhibits more significant improvement compared to EWC. From Figure 3.5(c), we could see the performance on multiple tasks (each column

shows the performance change of one task along the learning process) is well maintained when a small number of tasks are learned and then decrease drastically when more tasks are learned. A similar phenomenon has been observed in the performance matrix of GEM. Among all the baselines except the joint training, TWP, which is specially designed for continual graph learning via preserving topology information, seems to be the best in maintaining the performance of existing tasks. From Figure 3.5(e), we observe that the performance of many tasks is maintained until the end of the learning on the task sequence (last row). Similarly, ER-GNN, which is also specially designed for CGL, exhibits significant improvement. However, ER-GNN is sensitive to the length of the task sequence. On a short task sequence (the upper part of Figure 3.5(g)), the performance can be well maintained, but when more tasks are learned (the lower part of Figure 3.5(g)), the performance decreases drastically. This trend can also be observed in the learning curve shown in Figure 3.3(a). This phenomenon results from the memory mechanism of ER-GNN. ER-GNN selectively stores several nodes from each task and replays them when learning new tasks. The replay of the data from each previous task pushes the model parameters toward the direction that is favorable for its task. When too many tasks are stored, it would be hard for the model to find a direction to update, especially when the model has been already trained to an optimum of a certain task. This kind of forgetting pattern is not detectable solely from the final AP and AF, which only show the average results of the final row of the performance matrix.

Similar patterns can also be found in the results obtained on Reddit-CL as shown in Figure 3.6. The difference is that Reddit-CL is less challenging and most methods are performing better than on CoraFull-CL. Besides, since the task sequence of Reddit-CL is shorter than CoraFull-CL, ER-GNN is performing relatively well through the entire sequence.

Among all the datasets, almost all baselines fail on Arxiv-CL. To further investigate this issue, we visualize the performance matrices of all baselines on Arxiv-CL, as shown in Figure 3.7. According to the results, the baselines perform well on each individual task (diagonal entries), and the failure comes from the severe forgetting issue. The results reveal that the task-wise interference in Arxiv-CL is strong. Specifically, Bare model, EWC, GEM, and LwF exhibit almost complete forgetting on previous tasks, and their performance on new tasks are

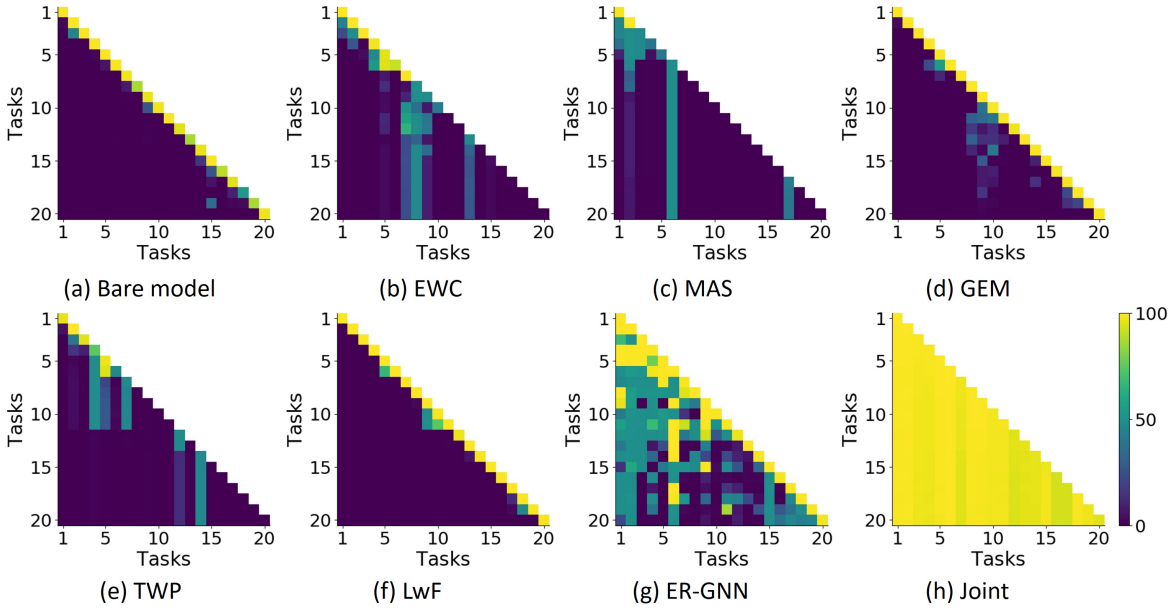


FIGURE 3.6. Visualization of the performance matrices of different methods on Reddit-CL under class-IL setting.

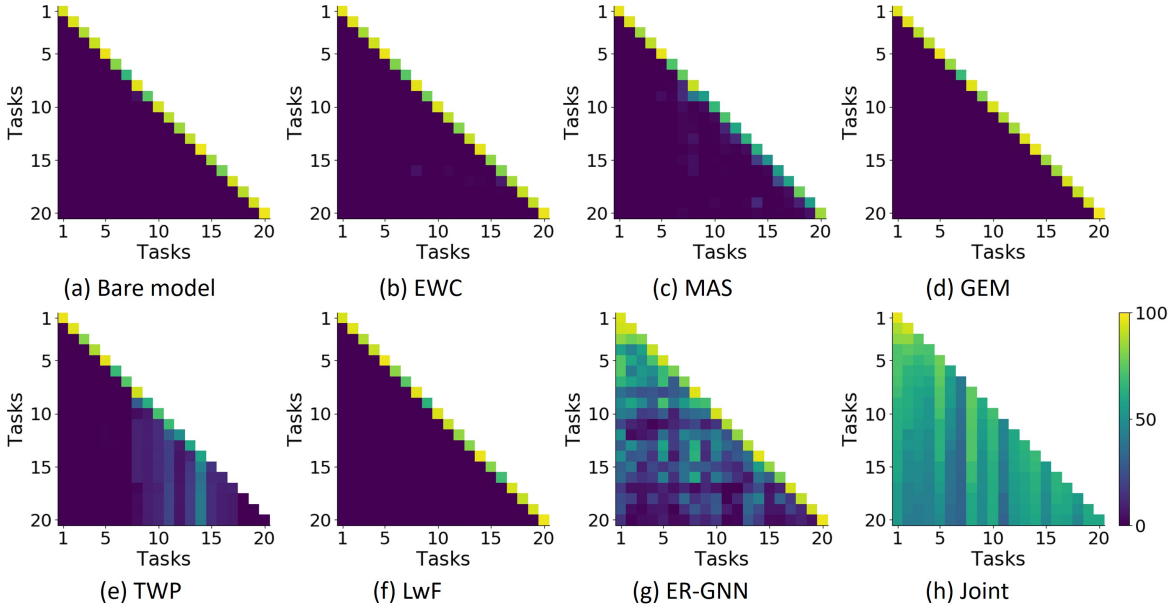


FIGURE 3.7. Visualization of the performance matrices of different methods on Arxiv-CL under class-IL setting.

better (diagonal entries). TWP, ERGNN, and MAS successfully preserve the performance on previous tasks to some extent, and their performance on the following tasks are lower. Specially, TWP preserves the performance on task 14 well, but fails in all the following

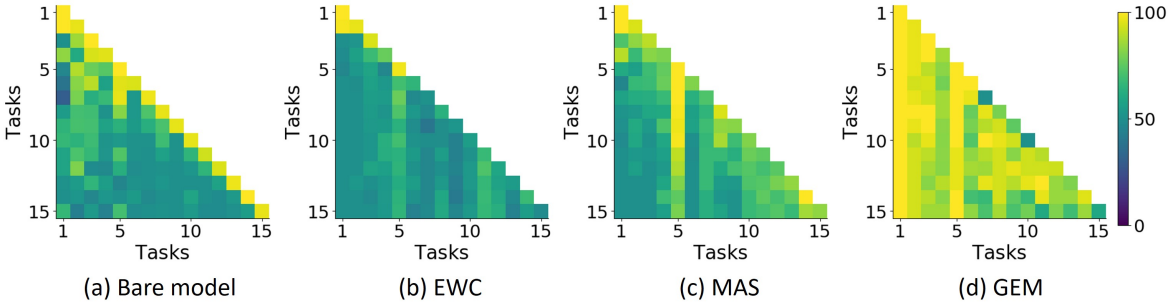


FIGURE 3.8. Visualization of the performance matrices of different methods on Aromaticity-CL under task-IL setting.

tasks. This strong task-wise interference is finally justified again in the performance matrix of the jointly trained model (Joint), which does not suffer from the forgetting issue. Although Joint maintains a balanced performance on all tasks, its diagonal entries have obviously lower values than the methods with severe forgetting. In a word, due to the strong task-wise interference, a method cannot simultaneously maintain the performance well on both old tasks and new tasks.

3.5.3 Further Analysis on the Model Performances on G-CGL tasks

To further demonstrate the learning dynamics on G-CGL tasks, we visualize the performance matrices of several methods in this subsection. The visualization is done for both task-IL and class-IL scenarios on the Aromaticity-CL dataset.

Figure 3.8 and 3.9 show the results on Aromaticity-CL under task-IL and class-IL scenarios, respectively. The bare model and joint training still follow similar patterns as it did in Section 2.2. In other words, the bare model forgets severely on previous tasks while joint training performs perfectly on all tasks. For the other methods under both task-IL and class-IL scenarios, we should focus more on the diagonal entries of their performance matrices. The diagonal entries of the performance matrices of the bare model still demonstrate that the bare model learns well on each new task. However, the diagonal entries of the regularization based methods like EWC, MAS, and TWP, decrease significantly along with the learning (from top to the bottom). This indicates that different tasks have significantly different distributions and

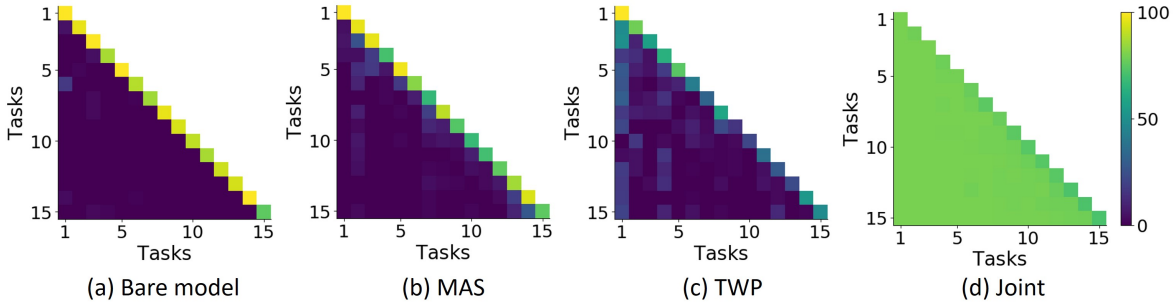


FIGURE 3.9. Visualization of the performance matrices of different methods on Aromaticity-CL under class-IL setting.

preserving the parameters of the previous tasks greatly hinders the learning of new tasks. This is also consistent with the finding that GEM often meets the 'no solution' error.

3.5.4 Additional Results with Less Training Data

Besides the train-validation-test split used in our major experiments, our implemented pipelines also support any other splittings through specifying the corresponding arguments (details can be found on our GitHub page). In the following, we provide the results obtained on both node-level and graph-level tasks with the splitting of train (20%), validation (40%), test (40%).

As shown in Table 3.16 and 3.17, the performance change of NCGL tasks and GCGL tasks with less training data are different. On NCGL tasks, most methods exhibit performance decrease, but some methods perform better. While in GCGL, almost all methods experience significant performance decrease. The reasons are two-fold. On the one hand, less training data may decrease the performance on single tasks. On the other hand, with less training data, the models adapt less to the new tasks, therefore the forgetting on previous tasks is less severe, which benefits the overall performance (AP and AF). The NCGL task sequences are longer than GCGL, and longer sequences bring more severe forgetting. Accordingly, the mitigation of the forgetting problem (by less training data) benefits the performance more in NCGL than GCGL. Therefore, the performance decreases more in GCGL than NCGL.

TABLE 3.16. Performance comparisons under task-IL without inter-task edges on different datasets with the splitting of train (20%), validation (40%), test (40%) (↑ higher means better).

C.L.T.	CoraFull-CL		Arxiv-CL		Reddit-CL		Products-CL	
	AP/% ↑	AF/% ↑	AP/% ↑	AF /% ↑	AP/% ↑	AF /% ↑	AP/% ↑	AF /% ↑
Bare model	53.2±1.2	-40.5±1.5	60.9±6.2	-27.8±5.9	80.0±7.4	-20.2±7.7	66.0±1.9	-26.8±2.1
EWC [44]	70.1±3.7	-20.9±4.1	68.4±4.5	-16.6±5.0	92.5±6.6	-7.1±6.9	90.3±0.7	-0.6±0.3
MAS [3]	90.1±1.1	-0.5±0.5	87.3±0.2	0.0±0.0	98.9±0.3	0.0±0.0	91.8±0.7	-0.5±0.1
GEM [59]	90.0±0.1	0.0±0.4	80.4±0.1	-4.0±0.3	99.3±0.0	0.0±0.1	87.6±0.9	-3.0±0.8
TWP [57]	86.9±0.5	-2.0±0.5	86.1±0.8	-1.7±0.7	91.2±3.6	-8.5±3.8	90.3±0.3	-0.6±0.3
LwF [52]	54.2±1.6	-39.7±1.6	68.6±4.9	-20.6±5.3	78.7±4.3	-21.7±4.6	66.6±1.8	-26.9±2.0
ER-GNN [129]	83.6±0.4	-6.7±0.5	89.2±0.1	5.6±0.5	98.8±0.1	-0.4±0.1	89.3±0.1	-2.4±0.2
Joint	91.9±0.5	-	88.9±0.4	-	99.4±0.0	-	91.2±0.8	-

TABLE 3.17. Performance comparisons on different graph-level prediction datasets with the splitting of train (20%), validation (40%), test (40%) (↑ higher means better).

C.L.T.	SIDER-tIL		Tox21-tIL		Aromaticity-CL		Aromaticity-CL	
	task-IL		task-IL		task-IL		class-IL	
	AP ↑	AF ↑	AP ↑	AF ↑	AP/% ↑	AF /% ↑	AP/% ↑	AF /% ↑
Bare model	0.532±0.007	0.028±0.016	0.645±0.022	0.119±0.015	52.0±1.4	0.1±1.2	4.3±1.4	-5.2±2.9
EWC [44]	0.503±0.012	0.003±0.006	0.602±0.021	0.028±0.027	52.0±1.4	0.1±1.2	3.9±0.8	-10.0±2.3
MAS [3]	0.518±0.010	0.014±0.006	0.630±0.022	0.092±0.029	58.8±2.0	5.0±2.2	3.8±1.0	-8.8±2.8
GEM [59]	0.578±0.006	0.072±0.014	0.685±0.007	0.183±0.025	70.6±2.2	18.8±3.4	10.9±1.7	2.1±3.5
TWP [57]	0.505±0.009	0.009±0.004	0.593±0.010	0.046±0.025	54.2±2.7	0.9±2.9	3.5±1.1	-8.7±2.6
LwF [52]	0.531±0.009	0.027±0.008	0.641±0.017	0.105±0.049	58.7±1.1	8.2±2.4	5.4±0.4	-8.4±0.9
Joint	0.575±0.009	-	0.678±0.017	-	69.4±1.0	-	35.4±3.9	-

3.5.5 Studies on the Number of Classes in each Task

In the experiments reported earlier, we set the number of classes in each task (K) as 2 so as to maximize the length of the task sequences and increase the learning difficulty. In this subsection. we further show the results obtained with multiple different Ks. As shown in Table 3.18, a clear pattern is that the continual learning performance of the models increase with the K. This phenomenon concerns two key factors. On the one hand, smaller K decreases the difficulty of each single task, which positively affects the performance. On the other hand, smaller K also increases the length of the task sequence and exacerbates the forgetting problem, which negatively affects the performance. Considering these two factors and the final results, we can conclude that the length of task sequence is the dominant factor determining the continual learning difficulty.

TABLE 3.18. Performance comparisons under class-IL on Arxiv-CL with different task splittings (↑ higher means better).

C.L.T.	$K=2$		$K=5$		$K=10$		$K=20$	
	AP/% ↑	AF/%↑	AP/% ↑	AF /% ↑	AP/% ↑	AF /% ↑	AP/% ↑	AF /% ↑
Bare model	4.9±0.0	-87.0±1.5	10.5±0.1	-77.5±0.5	16.4±0.2	-63.9±0.6	26.4±0.3	-47.3±0.9
EWC [44]	4.9±0.0	-88.9±0.3	9.4±0.1	-73.7±1.1	15.7±0.3	-62.8±0.7	24.8±0.3	-47.5±0.6
MAS [3]	4.9±0.0	-86.8±0.6	10.3±0.2	-77.5±0.6	16.5±0.3	-64.0±0.5	26.3±0.6	-47.5±0.7
GEM [59]	4.8±0.5	-87.8±0.2	10.7±0.1	-81.5±0.3	18.2±0.2	-70.6±0.5	31.3±0.1	-58.5±0.2
TWP [57]	4.9±0.0	-89.0±0.4	8.3±0.4	-66.1±1.3	14.0±0.4	-57.6±1.5	22.0±0.4	-47.6±0.5
LwF [52]	4.9±0.0	-87.9±1.0	10.5±0.1	-78.9±0.3	17.4±0.3	-66.2±0.5	28.6±0.1	-52.6±0.6
ER-GNN [129]	30.3±1.5	-54.0±1.3	10.9±0.2	-77.5±0.5	19.8±1.2	-59.9±1.3	31.6±0.6	-34.8±1.3
Joint	46.4±1.4	-	47.7±0.3	-	45.2±0.7	-	43.6±0.3	-

3.5.6 Benchmark Usages & Implementing New Methods with CGLB

The tutorial on using CGLB and repeating the results reported are provided on our Github page https://github.com/QueuQ/CGLB with examples.

Our pipelines for both node-level and graph-level tasks are highly modularized to facilitate the implementation and evaluation of new continual graph learning methods on CGLB.

We exemplify the implementation of node-level methods, and the generalization to graph-level methods is straightforward. The newly implemented method should be contained in a python script file under the directory *CGLB/NCGL/Baselines*. Suppose we are implementing a method named *A*, then an *CGLB/NCGL/Baselines/A_model.py* containing the implementation of the method should be created. The implementation is flexible as long as it satisfies the input format. Specifically, the python class of the new method should contain an *observe()* function for model training on a single task, whose input include the task configurations and the data. Details on the input format could be found in any *xxx_model.py* file under the directory *CGLB/NCGL/Baselines*.

3.5.7 Extension of Continual Learning to Scene Graph Research

3.5.7.1 Introduction

In the previous chapters, we mainly focus on the commonly studied graph types including different network data and molecule graphs. In this chapter, we further explore the application of continual learning tasks to scene graph related tasks. Different from the CGL research in the previous chapters, research on scene graphs are not conducted on constructed graphs. Instead, they focus on generating and refining of scene graphs from visual data. One important problem in scene graph research is how to properly learn the relationships in with limited labels. For example, [20] proposed a method to generate labels for the unlabelled images with a limited set of labeled examples. In this way, the rare relationships are assigned with more training data.

Based on this framework, a natural problem is: New relationship types may be encountered by the model gradually. At the beginning of the emergence of each new relationship, there will be a limited number of labeled examples available, which can be solved by technique proposed by [20]. But when multiple new relationships emerge sequentially, the model might experience catastrophic forgetting problem if the different relationships follow significantly different distributions. Therefore, in this chapter, we first study whether the forgetting behavior in the problem, and then adapt several continual learning techniques to this problem and investigate their effectiveness.

3.5.7.2 Method

Briefly speaking, the method for scene graph prediction with limited labels (SGPLL) [20] is designed to first extract image agnostic features (categorical and spatial features), and then adopt clustering based approach to label the unlabelled relations based on the limited number of relation labels. In our study, we construct a sequence of 5 tasks, each of which contains 4 unique relation types. In each task, the model will be trained to label the unlabelled data with the available labeled data of the relation types in this task. We implement 3 representative

TABLE 3.19. Performance comparisons between SGPLL with different continual learning baselines.

C.L.T.	AM/%	FM/%
Fine tune	56.4±9.1	-8.4±5.4
Memory Replay	58.3±8.5	-6.2±4.7
Joint Training	60.4±5.6	-

continual learning techniques including fine-tune, memory replay, and joint training. Fine-tune refers to directly fine-tune the model for each incoming new task, and can reflect the severeness of the forgetting problem. Memory-replay will train the model with both new data and buffered data when each new task comes, and is one the the most effective continual learning technique currently. Joint training, which jointly train the model on all tasks, is the upper bound on the model performance.

3.5.7.3 Results

In Table 3.19, the first row demonstrates that the forgetting phenomenon is significant in this problem, indicating that the distribution of the image agnostic features of different relation types are non negligible. Second, the second row indicates that incorporate memory buffer can significantly alleviate the forgetting issue. Finally, joint training performs the best. All of the three methods investigated in this chapter can actually be viewed as memory replay based method with different memory budgets. Fine-tune corresponds to a budget of 0, while joint training equals storing all previous data in the buffer. In a word, the experiments in this section demonstrate that the catastrophic forgetting problem also exist in the field of scene graph research, and can also be handled with general continual learning techniques.

3.6 Conclusion

In this benchmark work, we systematically analyzed the setup of continual graph learning and proposed the CGLB to thoroughly and fairly assess state-of-the-art methods from different perspectives, *i.e.*, N-CGL and G-CGL under both task-IL and class-IL scenarios. Besides, we

also tested several representative existing techniques on CGLB with detailed analysis with the model performance. By providing the CGLB as well as the toolkit for training, evaluation, and visualization, we greatly lower the barrier to explore CGL problem. In the future, we will also keep adopting new datasets and new benchmark results into CGLB. We will also pursue more practical settings for CGL.

CHAPTER 4

Hierarchical Prototype Networks for Continual Graph Representation Learning

---

4.1 Introduction

In this chapter, we introduce our first technical contribution, Hierarchical Prototype Networks (HPNs), which is based on both parameter isolation and memory-replay. To begin with, graph representation learning aims to pursue a meaningful vector representation of each node so as to facilitate downstream applications such as node classification, link prediction, *etc*. Traditional methods are developed based on graph statistics [71] or hand-crafted features [8, 53]. Recently, a great amount of attention has been paid to graph neural networks (GNNs), such as graph convolutional network (GCNs) [43], GraphSAGE [35], Graph Attention Networks (GATs) [87], and their extensions [105, 18, 134, 51, 19, 95]. This is because they can jointly consider the feature and topological information of each node. Most of these approaches, however, focus on static graphs and cannot generalize to the case when new categories of nodes are emerging.

In many real world applications, different categories of nodes and their associated edges (in the form of subgraphs) are often continuously emerging in existing graphs. For instance, in a citation network [79, 93, 67], papers describing new research areas will gradually appear in the citation graph; in a co-purchasing network such as Amazon [9], new types of products will continuously be added to the graph. Given these facts, how to incorporate the feature and topological information of new nodes in a continuous and effective manner such that performance over existing nodes is uninterrupted is a critical problem to investigate,

especially when graphs are relatively large and retraining a new model over an entire graph is computationally expensive.

To address this issue, a few attempts have been made to tailor the existing continual learning techniques to graph data. For instance, adopting the memory replay based approaches, Zhou et al. [129] proposed to store a set of representative experience nodes in a buffer and replay them along with new tasks (categories) to prevent forgetting existing tasks (categories). The buffer, however, only stores node features and ignores the topological information of graphs. Inspired by the regularization based methods, Liu et al. [57] developed topology-aware weight preserving (TWP) that can preserve the topological information of existing graphs. However, its design hinders the capability of learning topology on new tasks (categories). More detailed introduction on these related works can be found in Literature Review in Chapter 2.

A desired learning system for continual graph representation learning is expected to continuously grasp knowledge from new categories of emerging nodes and capture their topological structures without interfering with the learned knowledge over existing graphs. Inspired by the fact that humans learn to recognize objects by forming prototypes in the brain [12, 23, 117] and can achieve extraordinary continual learning capability, we present a completely novel framework, *i.e.*, Hierarchical Prototype Networks (HPNs), to continuously extract different levels of abstract knowledge (in the form of prototypes) from graph data such that new knowledge will be accommodated while earlier experience can still be well retained. Within this framework, representation learning is simultaneously conducted to avoid catastrophic forgetting, instead of considering these two objectives separately. Although learning independent prototypes for different tasks is appealing for preventing forgetting, it may incur unbounded memory consumption with an unlimited number of new tasks. Therefore, we propose to denote each node with a composition of basic prototypes so that nodes from different tasks and their associated relational structures are represented with compositions of a limited set of basic prototypes. Take the social network as an example, if a node (person) falls into certain category, it can be decomposed into basic atomic characteristics belonging to a set of attributes (*e.g.*, gender, nationality, hobby, *etc.*,) and the relationship between a pair of nodes can also be categorized into different basic types (*e.g.*, friends, coworkers, *etc.*).

Inspired by these facts, we first develop the Atomic Feature Extractors (AFEs) to decompose each node into two sets of atomic embeddings, *i.e.*, atomic node embeddings which encode the node feature information and atomic structure embeddings which encode its relations to neighboring nodes within multi-hop. Next, we present Hierarchical Prototype Networks to adaptively select, compose, and store representative embeddings with three levels of prototypes, *i.e.*, atomic-level, node-level, and class-level. Given a new node, only the relevant AFEs and prototypes in each level will be activated and refined, while others are uninterrupted.

The memory efficiency and the continual learning capability of the proposed HPNs are theoretically validated. First, adopting concepts from geometry and coding theory, we show the equivalence between the prototypes of HPNs and the spherical codes, thereby deriving the upper bound of the number of prototypes. This indicates that our memory consumption is bounded regardless of the number of tasks encountered. Second, since the forgetting problem can be formalized as the prediction drift of previous data after learning new tasks, we derive the conditions under which the learning on new tasks will not alter the predictions of previous data, *i.e.* forgetting is eliminated.

The theoretical merits of the proposed HPNs are justified by experiments on five public datasets. First, the performance of HPNs not only achieves the state-of-the-art, but is also comparable to or better than the jointly trained model (set as the upper bound of continual learning models). Second, the HPNs are memory efficient. For instance, on OGB-Products dataset containing more than 2 million nodes and 47 categories of nodes, HPNs achieve around 80% accuracy with only thousands of parameters.

4.2 Hierarchical Prototype Networks

In this section, we first state the problem we aim to study and the notations. Then we present Hierarchical Prototype Networks (HPNs) that consist of two core modules, *i.e.*, Atomic Feature Extractor (AFEs) and Hierarchical Prototypes (HPs), as shown in Figure 4.1. AFEs serve to extract a set of atomic features from the given graph, and the HPs aim to select, compose, and store the representative features in the form of different levels of prototypes.

During the training stage, each node will only refine the relevant AFEs and prototypes of the model without interfering with the irrelevant parts (*i.e.*, to avoid catastrophic forgetting). In the test stage, the model will activate the relevant AFEs and prototypes to perform the inference.

4.2.1 Problem Statement and Notations

We study continual learning on graphs that have new categories of nodes and associated edges (in the form of subgraphs) emerging in a continuous manner. In the context of continual learning, assuming we have a sequence of p tasks $\{\mathcal{T}^s|s=1,...,p\}$, in which each task $\mathcal{T}^s$ aims to learn a satisfied representation for a new subgraph $\mathcal{G}_s$ consisting of nodes belonging to some new categories. A desired model should maintain its performance on all previous tasks after being successively trained on the sequence of p tasks from $\mathcal{T}^1$ to $\mathcal{T}^p$.

For simplicity, we omit the subscripts in this section. Full notations will be used in the theoretical analysis. Each graph $\mathcal{G}$ consists of a node set $\mathbb{V}=\{v_i|i=1,...,N\}$ with N nodes and an edge set $\mathbb{E}=\{(v_i,v_j)\}$ denoting the connections of nodes in $\mathbb{V}$. Each node v_i can be represented as a feature vector $\mathbf{x}(v_i)\in\mathbb{R}^{d_v}$ that encodes node attributes, *e.g.*, gender, nationality, hobby, *etc*. The set of l-hop neighboring nodes of v_i is defined as $\mathcal{N}^l(v_i)$, with $\mathcal{N}^0(v_i)=\{v_i\}$.

4.2.2 Atomic Feature Extractors

As mentioned in the Introduction, to handle unlimited number of potential new categories of nodes with a limited memory consumption, we represent each node as a composition of basic prototypes selected from a limited set. To construct or refine these basic prototypes, given the input graph, we first need to obtain basic features with feature extractors. Specifically, we develop Atomic Feature Extractors (AFEs) to consider two different sets of atomic embeddings, *i.e.*, atomic node embeddings which encode the target node features and atomic structure embeddings that encode its relations to multi-hop neighbors. To ensure these generated basic prototypes are capable of encoding low-level features that can be shared

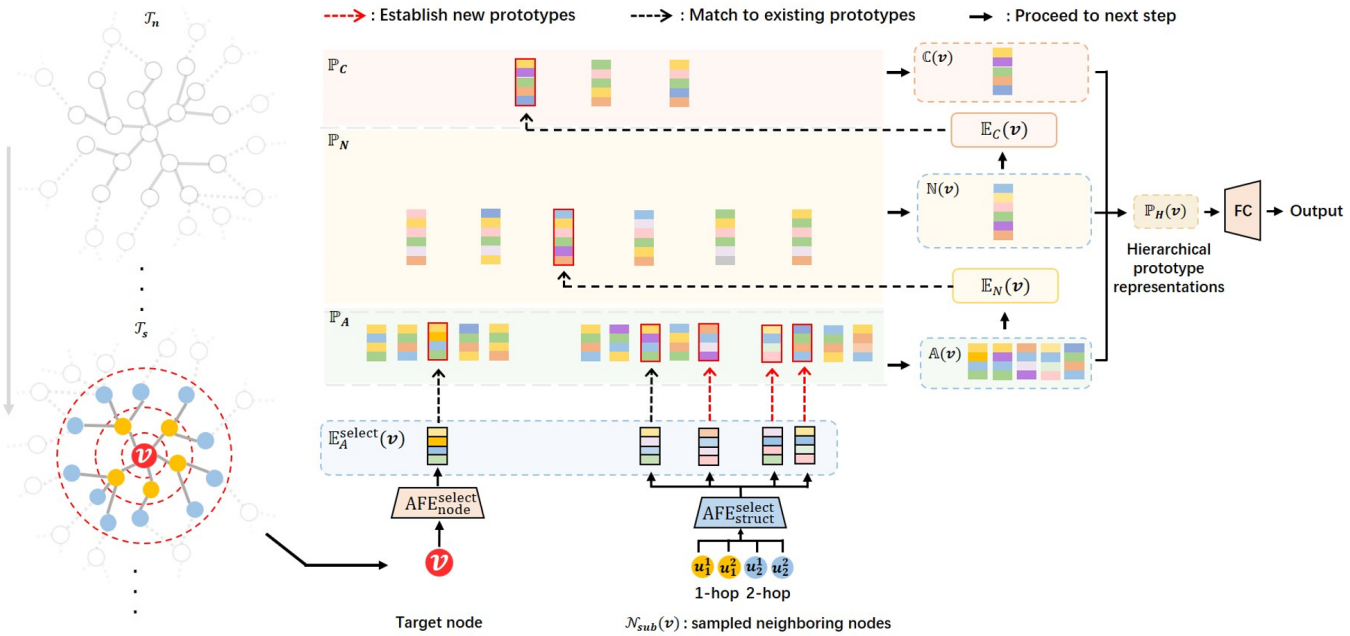


FIGURE 4.1. The framework of HPNs. On the left, subgraphs of different tasks come in sequentially, with the current task denoted as $\mathcal{T}_s$. Given a node v, u_k^j denotes the j-th sampled node from k-hop neighbors. In the middle, node v and the sampled neighbors ($\mathcal{N}_{sub}(v)$) are fed into the selected AFEs ($\text{AFE}_{\text{node}}^{\text{select}}$ or $\text{AFE}_{\text{struct}}^{\text{select}}$) to get atomic embeddings ($\mathbb{E}_A^{\text{select}}(v)$), which are either matched to existing A-prototypes ($\mathbb{P}_A$) denoted with black dashed arrows or used to establish new A-prototypes (red dashed arrows). The selected A-prototypes $\mathbb{A}(v)$ are first mapped to node- and class-level embeddings ($\mathbb{E}_N(v)$ and $\mathbb{E}_C(v)$), which are further matched to a N- and a C-prototypes ($\mathbb{P}_N$ and $\mathbb{P}_C$) to get the node- and class-level prototype representations ($\mathbb{N}(v)$ and $\mathbb{C}(v)$). Finally, $\mathbb{A}(v)$, $\mathbb{N}(v)$ and $\mathbb{C}(v)$ form the hierarchical prototype representation ($\mathbb{P}_H(v)$), which is fed into the classifier to perform node classification.

across different tasks, we avoid deep structures in the AFEs and formulate them as learnable linear transformations: $\text{AFE}_{\text{node}} = \{\mathbf{A}_m \in \mathbb{R}^{d_v \times d_a} | m \in \{1, ..., l_a\}\}$ and $\text{AFE}_{\text{struct}} = \{\mathbf{R}_n \in \mathbb{R}^{d_v \times d_r} | n \in \{1, ..., l_r\}\}$ where $\mathbf{A}_m$ and $\mathbf{R}_n$ are real matrices to encode atomic node and structure information, respectively. Each matrix $\mathbf{A}_m$ or $\mathbf{R}_n$ corresponds to a certain type of features, and l_a, l_r denote the number of matrices in AFE_{node} and $\text{AFE}_{\text{struct}}$, respectively. Given a node v, a set of atomic node embeddings is obtained by applying AFE_{node} to the feature vector $\mathbf{x}(v)$:

$$\mathbb{E}_A^{\text{node}}(v) = \{\mathbf{x}^T(v)\mathbf{A}_m | \mathbf{A}_m \in \text{AFE}_{\text{node}}\}. \tag{4.1}$$

To obtain atomic structure embeddings, the multi-hop neighboring nodes are considered to encode the relationship between node v and neighboring nodes within a fixed number of hops. Since the number of neighboring nodes varies from node to node, for computation efficiency in the following procedure, we uniformly sample a fixed number of vertices from 1-hop up to h-hop neighborhood. The sampled neighborhood set is denoted as, $\mathcal{N}_{sub}(v) \subseteq \bigcup_{l\in\{1,\dots,h\}} \mathcal{N}^l(v)$. Then these selected neighboring nodes are embedded via matrices in $\mathrm{AFE}_{\text{struct}}$ to encode different types of interactions between the target node v and its neighbors:

$$\mathbb{E}_A^{\text{struct}}(v) = \{\mathbf{x}^T(u)\mathbf{R}_m | \mathbf{R}_m \in \mathrm{AFE}_{\text{struct}}, u \in \mathcal{N}_{sub}\}. \tag{4.2}$$

Above all, with $\mathrm{AFE}_{\text{node}}$ and $\mathrm{AFE}_{\text{struct}}$, the attribute and structure information of each node v is encoded in the two set $\mathbb{E}_A^{\text{node}}(v)$ and $\mathbb{E}_A^{\text{struct}}(v)$. For convenience of notation, we denote the complete atomic feature set of a node v as the union of the node atomic embedding set $\mathrm{AFE}_{\text{node}}$ and the structure embedding set $\mathrm{AFE}_{\text{struct}}$, *i.e.*,

$$\mathbb{E}_A(v) = \mathbb{E}_A^{\text{node}}(v) \cup \mathbb{E}_A^{\text{struct}}(v). \tag{4.3}$$

Note that different $\mathbf{A}_i$ and $\mathbf{R}_i$ are designed to generate different types of atomic features. Therefore, we impose a divergence loss on AFEs to ensure they are uncorrelated with each other and thus can map features to different subspaces:

$$\mathcal{L}_{div} = \sum_{m=1}^{l_a} \sum_{\substack{n=1 \\ n\neq m}}^{l_a} \mathbf{A}_m^T \mathbf{A}_n + \sum_{m=1}^{l_r} \sum_{\substack{n=1 \\ n\neq m}}^{l_r} \mathbf{R}_m^T \mathbf{R}_n. \tag{4.4}$$

4.2.3 Hierarchical Prototype Networks

Based on the atomic embeddings of the incoming subgraph, HPNs will distill and store the representative features in form of different levels of prototypes to memorize the current task, as shown in Figure 4.1. This is achieved by refining existing prototypes if the input subgraph only contains features observed before and creating new prototypes if new features are encountered. Specifically, HPNs will produce three levels of prototypes. First, the atomic-level prototypes (A-prototypes) are directly refined or created with the atomic embedding set, and they denote the prototypical basic features of nodes (*e.g.*, gender, nationality, and social

relation of people in a social network). Since the A-prototypes only describe low-level features of a node, which is similar to the output from the initial layers of a deep neural network, we further develop node-level (N-prototypes) and class-level prototypes (C-prototypes) to capture higher level information of each node. The N-prototypes are generated by associating the A-prototypes with each node to describe the node as a whole. The C-prototypes are generated from the N-prototypes to describe common features shared by a group of similar nodes. From atomic-level to class-level, the prototypes denote abstract knowledge of the subgraph at different scales which is analog to the feature maps of convolutional neural networks at different layers.

Next, we introduce when and how HPNs will refine existing prototypes or establish new ones with the atomic embeddings. As mentioned earlier, for each task that contains certain categories of nodes, only the most relevant AFEs and prototypes will be activated while others remain uninterrupted to avoid forgetting. In other words, for each node, we will choose a subset of $\mathbb{E}_A(v)$ to refine the prototypes. Specifically, as shown in Figure 4.1, given a node from an incoming subgraph, The AFEs whose atomic embeddings are close enough to existing A-prototypes are chosen as the relevant ones. The closeness (distance) between embeddings and prototypes is measured with cosine similarity for two considerations. The first consideration is the numerical stability. Since we are going to develop an objective (Equation 4.8) to minimize the distance between the matched embeddings and prototypes, if the embeddings and the prototypes are not normalized, Equation 4.8 becomes maximizing the inner product between two unnormalized vectors. Unlike the cosine similarity that only adjusts the angle between two vectors and reaches the upper bound of 1 when the angle is 0, the inner product is unbounded and maximizing the inner product causes the vector lengths to increase to infinity, leading to the collapse of the model. Second, to guarantee an upper bound on the memory consumption, we restrict the prototypes on a unit sphere via normalization, so that only a limited number of prototypes will be created (Theorem 1). Due to the normalization, cosine distance is a natural choice as the distance metric.

Formally, we first obtain $\mathbb{E}_A^{\text{node}}(v)$ and $\mathbb{E}_A^{\text{struct}}(v)$ via Eq. (4.1) and Eq. (4.2), respectively. Then, we calculate the maximum cosine similarity between atomic embeddings of each AFE ($\mathbf{e}_i$)

and the A-prototypes as:

$$\mathrm{SimMAX}_i^{\mathrm{id}} = \max_{\mathbf{p}}\left(\frac{\mathbf{e}_i^T\mathbf{p}}{\|\mathbf{e}_i\|_2\|\mathbf{p}\|_2}\right), \mathbf{e}_i \in \mathbb{E}_A^{\mathrm{id}}(v), \mathbf{p} \in \mathbb{P}_A, \tag{4.5}$$

where id $\in \{\text{node}, \text{struct}\}$, i ranges from 1 to l_a (or l_r), and $\mathbb{P}_A$ is the atomic prototype set containing all A-prototypes. After that, we sort the AFEs in a descending order according to $\mathrm{SimMAX}_i^{\mathrm{id}}$ as $\mathrm{AFE}_{\text{node}}^{\text{sort}} = \{\mathbf{A}_{m'} \in \mathbb{R}^{d_v \times d_a} | m' \in \{1, ..., l_a\}\}$ and $\mathrm{AFE}_{\text{struct}}^{\text{sort}} = \{\mathbf{R}_{n'} \in \mathbb{R}^{d_v \times d_r} | n' \in \{1, ..., l_r\}\}$. Finally, we select the top l'_a and top l'_r ranked AFEs from these two sets as $\mathrm{AFE}_{\text{node}}^{\text{select}}$ and $\mathrm{AFE}_{\text{struct}}^{\text{select}}$, respectively. l'_a and l'_r are fixed hyperparameters with $l'_a \leq l_a$ and $l'_r \leq l_r$. Finally, the atomic embeddings generated by these selected AFEs are denoted as $\mathbb{E}_A^{\text{select}}(v)$, and we have $\mathbb{E}_A^{\text{select}}(v) \subset \mathbb{E}_A(v)$.

The $\mathbb{E}_A^{\text{select}}(v)$ is then used for refining existing prototypes or establishing new ones. To determine which embeddings contain observed features and which contain new features, a matching process is first conducted between the $\mathbb{E}_A^{\text{select}}(v)$ and $\mathbb{P}_A$. Formally, we measure the cosine similarity between elements in $\mathbb{E}_A^{\text{select}}(v)$ and elements in $\mathbb{P}_A$ as

$$\mathrm{Sim}_{E\to A}(v) = \{\frac{\mathbf{e}_i^T\mathbf{p}}{\|\mathbf{e}_i\|_2\|\mathbf{p}\|_2} | \mathbf{e}_i \in \mathbb{E}_A^{\text{select}}(v), \mathbf{p} \in \mathbb{P}_A\}. \tag{4.6}$$

Given $\mathrm{Sim}_{E\to A}(v)$, the atomic embeddings with cosine similarity (not less than a certain threshold t_A) to at least one existing A-prototype are regarded as containing observed features, since they are close to existing prototypes, *i.e.*,

$$\mathbb{E}_{old}(v) = \{\mathbf{e}_i | \quad \exists \mathbf{p} \in \mathbb{P}_A \quad s.t. \quad \frac{\mathbf{e}_i^T\mathbf{p}}{\|\mathbf{e}_i\|_2\|\mathbf{p}\|_2} \geqslant t_A\}. \tag{4.7}$$

$\mathbb{E}_{old}(v)$ is then used to refine $\mathbb{P}_A$. To this end, a distance loss $\mathcal{L}_{dis}$ is computed to enhance the cosine similarity between each $\mathbf{e}_i \in \mathbb{E}_{old}(v)$ and its corresponding A-prototype $\mathbf{p}_i \in \mathbb{P}_A$, *i.e.*,

$$\mathcal{L}_{dis} = -\sum_{\mathbf{e}_i \in \mathbb{E}_{old}(v)} \frac{\mathbf{e}_i^T\mathbf{p}_i}{\|\mathbf{e}_i\|_2\|\mathbf{p}_i\|_2} \tag{4.8}$$

By minimizing $\mathcal{L}_{dis}$, not only the existing A-prototypes in $\mathbb{P}_A$ will get refined, the atomic embeddings will also be closer to 'standard' A-prototypes.

Next, we discuss how to deal with the atomic embeddings that are not close to any existing prototypes, *i.e.*, $\mathbb{E}_{new}(v) = \mathbb{E}_A^{\text{select}}(v) \backslash \mathbb{E}_{old}(v)$ or $\mathbb{E}_{new}(v) = \{\mathbf{e}_i | \quad \forall \mathbf{p} \in \mathbb{P}_A, \quad \frac{\mathbf{e}_i^T \mathbf{p}}{\|\mathbf{e}_i\|_2 \|\mathbf{p}\|_2} < t_A\}$. Specifically, during the first task, $\mathbb{P}_A$ is initialized with one random A-prototype. The matching process is still the same, and the only difference is that none of the atomic embeddings can be matched to the existing A-prototype. Therefore, all atomic embeddings may be candidates for creating new A-prototypes, as explained below.

Contrary to $\mathbb{E}_{old}(v)$, atomic embeddings in $\mathbb{E}_{new}(v)$ are regarded as new atomic features of the corresponding AFEs and should be accommodated with new prototypes. Considering that very similar embeddings may exist in $\mathbb{E}_{new}(v)$ and cause HPNs to create redundant prototypes, we first filter $\mathbb{E}_{new}(v)$ to obtain $\mathbb{E}'_{new}(v)$ which only contains the distinctive ones such that

$$\forall \mathbf{e}_i, \mathbf{e}_j \in \mathbb{E}'_{new}(v), \frac{\mathbf{e}_i^T \mathbf{e}_j}{\|\mathbf{e}_i\|_2 \|\mathbf{e}_j\|_2} < t_A. \tag{4.9}$$

In this way, the embeddings in $\mathbb{E}'_{new}(v)$ only contain new features, they are included into $\mathbb{P}_A$ as new A-prototypes. Formally, the updated $\mathbb{P}_A$ is the union of the previous A-prototypes $\mathbb{P}_A$ and new A-prototypes $\mathbb{E}'_{new}(v)$

$$\mathbb{P}_A = \mathbb{P}_A \cup \mathbb{E}'_{new}(v). \tag{4.10}$$

After updating $\mathbb{P}_A$, $\text{Sim}_{E \to A}(v)$ will also be updated according to Eq. (4.6). Then, every element in $\text{Sim}_{E \to A}(v)$ would be not less than t_A, *i.e.*, each element in $\mathbb{E}_A^{\text{select}}(v)$ is matched to an A-prototype with cosine similarity larger than the threshold t_A. These matched A-prototypes are denoted as $\mathbb{A}(v)$, which represents the basic features of node v as shown in Figure 4.1.

As mentioned earlier, besides $\mathbb{A}(v)$, higher level representations are also beneficial for comprehensively describing a node from multiple granularities. Therefore, we further map $\mathbb{A}(v)$ to high level prototypes so as to obtain hierarchical prototype representations, Firstly, $\mathbb{A}(v)$ is mapped to an N-prototype to describe v as a whole. Specifically, the vectors in $\mathbb{A}(v)$ are concatenated as $\mathbf{a}_1 \oplus \cdots \oplus \mathbf{a}_{l'_a + l'_r}$ ($\mathbf{a}_i \in \mathbb{A}(v)$), and then fed into a full connected layer $\text{FC}_{A \to N}(\cdot)$ to generate node-level embedding set $\mathbb{E}_N(v) = \{\text{FC}_{A \to N}(\mathbf{a}_1 \oplus \cdots \oplus \mathbf{a}_{l'_a + l'_r})\}$. Here, for notation consistency with the aforementioned atomic embedding set, we also formulate the

node-level embeddings as a set $\mathbb{E}_N(v)$ although it contains only one element. With $\mathbb{E}_N(v)$, the following procedure is to form a N-prototype representation $\mathbb{N}(v)$ for node v, which is same as the process to form $\mathbb{A}(v)$ at atomic level except that the threshold is set as t_N, instead of t_A. Finally, to enrich the node representation with the common features shared with similar nodes, we develop the C-prototypes. The generation of C-prototypes is similar to the N-prototypes. We first obtain a class-level embedding set $\mathbb{E}_C(v)$ by applying another fully connected layer $\text{FC}_{N\to C}$ to the element in $\mathbb{N}(v)$. Then, we match $\mathbb{E}_C(v)$ to existing C-prototypes or establish new ones. Different from N-prototypes, the threshold used here is t_C, which is typically larger than t_A and t_N. In this way, each C-prototype will be matched to embeddings within a large space, and therefore representing the common features shared by a wide range of similar nodes. Again, for notation convenience, we derive $\mathbb{P}_H(v)$ to represent the hierarchical prototype representations of the target node v (as shown in Figure 4.1), which is the union of $\mathbb{A}(v)$, $\mathbb{N}(v)$, and $\mathbb{C}(v)$:

$$\mathbb{P}_H(v) = \mathbb{A}(v) \cup \mathbb{N}(v) \cup \mathbb{C}(v). \tag{4.11}$$

Algorithm 1 Learning Procedure for HPNs.

Input: Task sequence: $\{\mathcal{T}_1, ..., \mathcal{T}_p\}$, HPNs
for each $\mathcal{T} \leftarrow 1$ **to** p **do**
    Get the data of the current task: $\mathbb{V}$, $\mathbb{E}$, $\mathbf{X}(\mathbb{V}) = \{\mathbf{x}(v)|v \in \mathbb{V}\}$.
    Select $\text{AFE}^{\text{select}}_{\text{node}}$ and $\text{AFE}^{\text{select}}_{\text{struct}}$.
    Compute $\mathcal{L}$ = HPNs($\mathbb{V}$,$\mathbf{X}(\mathbb{V})$,$\mathbb{E}$).
    Optimize $\mathcal{L}$.
end for each
Output:updated HPNs

4.2.4 Learning Objective

The obtained hierarchical prototypes of each node are first concatenated into a unified vector and then forward through a fully connected layer FC to obtain a c (the number of classes) dimensional feature vector, *i.e.*, $\text{FC}(\mathbf{h}_1 \oplus \cdots \oplus \mathbf{h}_{l'_a+l'_r+2}), \forall \mathbf{h}_i \in \mathbb{P}_H(v)$. In this work, we aim to perform node classification. Therefore, based on the c dimensional feature vector and the softmax function $\sigma(\cdot)$, we can estimate the label with $\hat{y}_c = \sigma(\text{FC}(\mathbf{h}_1 \oplus \cdots \oplus \mathbf{h}_{l'_a+l'_r+2}))_c$

where c is the index of class. To perform node classification, with the output predictions $\hat{y}_c$ and the target label $y_c \in \{1, 2, ..., c\}$, the corresponding classification loss is given by

$$\mathcal{L}_{cls} = \sum_{c=1}^{C} -y_c \log(\hat{y_c}), \tag{4.12}$$

which is essentially the cross entropy loss function. As a discrepancy metric between probability distributions, cross entropy loss is generally preferred for classification tasks, because both the predictions $\{\hat{y}_c | c = 1, ..., C\}$ and the labels are discrete probability distributions. Note that besides node classification, $\mathbb{P}_H(v)$ may also be used for other tasks based on different objective functions. In this work, we focus on node classification and the overall loss of HPNs is:

$$\mathcal{L} = \mathcal{L}_{cls} + \alpha \cdot \mathcal{L}_{div} + \beta \cdot \mathcal{L}_{dis}, \tag{4.13}$$

where $\alpha \geq 0$ and $\beta \geq 0$ are hyperparameters to balance the two auxiliary losses.

4.2.5 Theoretical Analysis

In this subsection, we provide the theoretical upper bound on the memory consumption and analyze how the model configuration would affect HPNs' capacity in dealing with different tasks. Both theoretical results are justified and analyzed in the experiments. Only the main results are provided here, while the detailed proof and analysis are given in Section 4.4.

We first formulate the upper bound on the numbers of prototypes. Due to the mechanism to create new prototypes for newly emerging knowledge extracted from the graph, the memory consumption will gradually increase. However, the normalization applied to the prototypes constrains the prototype space, and thereby adds an upper bound on the memory consumption. This can be intuitively understood as the only limited number of points can be scattered on a n-dimensional hypersphere if we force the distance between each pair of points to be larger than a threshold. To formally formulate this, we first borrow the definition of spherical code from geometry and coding theory.

DEFINITION 1 (Spherical code). *A spherical code $S(n, N, t)$, with parameters (n, N, t) is defined as the set of N points on the unit hypersphere in an n-dimension space for which the dot product of unit vectors from the origin to any two points is less than or equal to t.*

In HPNs, the prototypes at different levels are normalized into unit vectors and can be viewed as spherical codes in their own space. Taking the A-prototypes as an example, given the dimension d_a and the threshold t_A, $\mathbb{P}_A$ is a spherical code $\mathbb{P}_A = S_A(d_a, N_A, 1 - t_A)$, where N_A is the cardinality of $\mathbb{P}_A$. Then the upper bound on the number of atom prototypes given d_a and t_A is equal to the maximal cardinality of $S(n, N, t)$ given n and t. Since the area of the n-dimensional unit sphere surface is limited, there exists a maximal N given a certain n and t, denoted as $\max_N S(d_a, N, 1 - t_A)$. However, finding $\max_N S(d_a, N, 1 - t_A)$ is a complex sphere packing problem, and there is not yet a general formulation of the maximal N for an arbitrary n. Therefore, given the number of two types of AFEs as l_a and l_r, we can formulate the upper bound on the numbers of different prototypes as:

THEOREM 1 (Upper bounds on numbers of prototypes). *Given the notations defined in HPNs, the upper bound on the number of A-prototypes n_a can be given by*

$$n_A \leqslant (l_a + l_r) \max_N S(d_a, N, 1 - t_A), \tag{4.14}$$

and the upper bounds on the number of N-prototypes and the C-prototypes are:

$$n_N \leqslant \max_N S(d_n, N, 1 - t_N) \tag{4.15}$$

$$n_C \leqslant \max_N S(d_c, N, 1 - t_C) \tag{4.16}$$

where $S(n, N, t)$ is the spherical code defined on a n dimensional hypersphere .

Theorem 1 provides an upper bound on the memory consumption of HPNs. If the upper bound of the number of prototypes is reached, the model is still able to learn new tasks. This is because the ability to learn new tasks depends on new combinations of the AFEs. The budget (*i.e.*, the limit) of the number of prototypes corresponds to the detail level of the basic properties extracted by AFEs. The larger budget is given to the number of prototypes, the more detailed representation can be generated for each node. In our experiments, we show

that the number of parameters for most baseline methods are even higher than this upper bound.

Besides memory consumption, the more important problem for a continual learning model is the capability to maintain memory on previously learned tasks. Based on our model design, we formulate this as: whether learning new tasks affect the representations the model generates for old task data. We give explicit definitions on tasks and task distances based on set theory (in Section 4.4), then construct a bound to indicate what configuration would the model have to ensure this capability.

THEOREM 2 (Task distance preserving). *For* HPNs *trained on consecutive tasks* $\mathcal{T}^p$ *and* $\mathcal{T}^{p+1}$. *If* $l_a d_a + l_r d_r \geqslant (l_r + 1)d_v$ *and* $\mathbf{W}$ *is column full rank, then as long as* $t_A < \lambda_{\min}(l_r+1)\mathbf{dist}(\mathbb{V}_p, \mathbb{V}_{p+1})$, *learning on* $\mathcal{T}^{p+1}$ *will not modify representations* HPNs *generate for data from* $\mathcal{T}^p$ *, i.e., catastrophic forgetting is avoided.*

The proof of Theorem 2 requires a number of definitions, lemmas, and corollaries, and the details are provided in Section 4.4. The theoretical results also provide insights into model implementations. In Theorem 2, λ_i is eigenvalues of the $\mathbf{W}^T\mathbf{W}$, where $\mathbf{W}$ denotes the matrix constructed via AFEs (details in Section 4.4). d_v, d_a and d_r are dimensions of data and two kinds of atomic embeddings. The bound in this theorem is not tight, since the tight bound would be dependent on the specific dataset properties. But this informs us that either the number of AFEs or the dimension of the prototypes has to be large enough to ensure that data from two tasks can be well separated in the representation space. Then, according to Theorem 1, the upper bound of the memory consumption is dependent on $S(d_a, N, t_A)$, $S(d_n, N, t_N)$, and $S(d_c, N, t_C)$. As $S(n, N, t)$ grows fast with n, we prefer larger number of AFEs with smaller prototype dimensions. We also empirically verify this in Section 4.3.6. Besides, the upper bound proposed in Theorem 1 is explicitly computed and compared to experimental results.

4.3 Experiments

In the experiments, we answer the following six questions: (1) Whether HPNs can outperform state-of-the-art approaches? (2) How does each component of HPNs contribute to its performance? (3) Whether HPNs can memorize previous tasks after learning each new task? (4) Are HPNs sensitive to the hyperparameters? (5) Whether the theoretical results can be empirically verified? (6) Whether the learned prototypes can be interpreted via visualization? All experiments are repeated on 5 different datasets.

TABLE 4.1. The detailed statistics of 5 datasets used in our experiments.

Dataset	Cora	Citeseer	Actor	OGB-Arxiv	OGB-Products
# nodes	2,708	3,327	7,600	169,343	2,449,029
# edges	5,429	4,732	33,544	1,166,243	61,859,140
# features	1,433	3,703	931	128	100
# classes	7	6	4	40	47
# tasks	3	3	2	20	23

4.3.1 Datasets

To assess the effectiveness of the proposed HPNs, we consider 5 public datasets which include 3 citation networks (Cora [79], Citeseer[79], OGB-Arxiv [93, 67]), 1 actor co-occurrence network (Actor) [69], and 1 product co-purchasing networks (OGB-Products [9]). The statistics of these 5 datasets are summarized in Table 4.1.

4.3.1.1 Citation networks

Cora contains 2708 documents, 5429 links denoting the citations among the documents. For training, 140 documents are selected with 20 examples for each class. The validation set contains 500 documents and the test set contains 1000 examples. In our continual learning setting, the first 6 classes are selected and grouped into 3 tasks (2 classes per task) in the original order. Citeseer contains 3312 documents and 4732 links. 20 documents per class are selected for training, 500 documents are selected for validation, and 1000 documents

are selected as the test set. For continual learning setting, the documents from 6 classes are grouped into 3 tasks with 2 classes per task in the original order. The Cora and Citeseer datasets can be downloaded via Cora&Citeseer.

The OGB-Arxiv dataset is collected in the Open Graph Benchmark OGB. It is a directed citation network between all Computer Science (CS) arXiv papers indexed by MAG [93]. Totally it contains 169,343 nodes and 1,166,243 edges. The dataset contains 40 classes. As the dataset is imbalanced and the numbers of examples in different classes differs significantly, directly grouping the classes into 2-class groups like the Cora and Citeseer will cause the tasks to be imbalanced. Therefore, we reordered the classes in an descending order according to the number of examples contained in each class, and then group the classes according to the new order. In this way, the number of examples contained in different classes of each task are arranged to be as balanced as possible. Specifically, the class indices of each task are: {(35, 12), (15, 21), (28, 30), (16, 24), (10, 34), (8, 4), (5, 2), (27, 26), (36, 19), (23, 31), (9, 37), (13, 3), (20, 39), (22, 6), (38, 33), (25, 11), (18, 1), (14, 7), (0, 17), (29, 32)}, where each tuple denotes a task consisting of two classes.

4.3.1.2 Actor co-occurrence network

The actor co-occurrence network is a subgraph of the film-director-actor-writer network [84]. Each node in this dataset corresponds to an author, and the edges between the nodes are co-occurrence on the same Wikipedia pages. The whole dataset contains 7600 nodes and 33544 edges. Each node is accompanied with a feature vector of 931 dimensions. For this dataset, we also constructed 2 tasks with 2 classes per task. The link to this dataset is Actor. The balanced splitting of the classes is {(0, 1), (2, 3)}.

4.3.1.3 Product co-purchasing network

OGB-Products is collected in the Open Graph Benchmark OGB, representing an Amazon product co-purchasing network link. It contains 2,449,029 nodes and 61,859,140 edges. Nodes represent products sold in Amazon, and edges indicate that the connected products are purchased together. In our experiments, we select 46 classes and omit the last class containing

TABLE 4.2. Performance comparisons between HPNs and baselines on 5 different datasets.

C.L.T.	Base	Cora		Citeseer		Actor		OGB-Arxiv		OGB-Products	
		AM/%	FM/%	AM/%	FM/%	AM/%	FM /%	AM/%	FM /%	AM/%	FM /%
None	GCN	63.5±1.9	-42.3±0.4	64.5±3.9	-7.7±1.6	43.6±3.6	-9.1±2.9	56.8±4.3	-19.8±3.2	45.2±5.6	-27.8±7.1
	GAT	71.9±3.8	-33.1±2.3	66.8±0.9	-19.6±0.3	53.1±2.7	-4.3±1.6	54.3±3.5	-21.7±4.6	44.9±6.9	-30.3±5.2
	GIN	68.3±2.3	-35.4±3.4	57.7±2.3	-36.4±0.3	45.5±2.3	-8.9±2.8	53.2±6.5	-23.5±8.1	43.1±7.4	-31.4±8.8
EWC [44]	GCN	63.1±1.2	-42.7±1.6	54.4±4.2	-30.3±0.9	44.3±1.1	-7.1±1.4	72.1±2.4	-9.1±1.9	66.7±0.5	-8.4±0.4
	GAT	72.2±1.5	-32.2±1.6	65.7±2.5	-19.7±2.3	54.2±2.5	-2.5±1.5	73.2±1.1	-10.8±2.1	67.9±1.0	-9.6±1.3
	GIN	69.6±2.6	-28.5±2.8	57.9±3.4	-36.3±2.4	47.6±2.1	-7.2±1.6	74.1±1.7	-8.3±2.0	67.3±2.3	-13.6±1.5
LwF [52]	GCN	76.1±1.4	-21.3±2.4	67.0±0.2	-8.3±2.7	49.7±2.5	-3.6±1.4	69.9±3.9	-12.1±2.8	66.3±2.5	-11.8±3.4
	GAT	70.8±2.8	-34.6±4.1	66.1±4.1	-18.9±1.5	52.8±2.7	-6.2±2.2	68.9±4.4	-13.6±3.3	65.1±4.1	-13.2±2.9
	GIN	74.1±2.7	-23.3±0.8	63.1±1.9	-16.5±2.2	49.7±2.6	-4.1±1.5	71.4±4.8	-15.9±5.6	65.9±4.0	-10.7±3.1
GEM [59]	GCN	75.7±3.0	-6.5±4.4	41.8±2.6	-31.9±1.4	52.7±3.1	+3.9±2.9	75.4±1.7	-13.6±0.5	71.3±1.7	-10.5±0.9
	GAT	69.8±3.0	-26.1±2.6	71.3±2.2	+9.0±1.5	54.3±3.6	-2.0±0.9	76.6±0.7	-11.3±0.4	70.4±0.8	-10.9±1.6
	GIN	80.2±3.3	-2.0±4.2	49.7±0.5	-24.5±0.9	45.2±2.8	-11.1±1.5	77.3 ±2.1	-11.2±1.6	76.5±3.3	-7.2±2.5
MAS [3]	GCN	65.5±1.9	-21.4±3.7	59.5±3.1	-0.1±2.4	50.7±2.4	-1.5±0.8	69.8±0.4	-18.8±0.9	62.0±1.1	-17.9±1.9
	GAT	84.7±0.7	-5.6±2.0	69.1±1.1	-4.8±3.3	53.7±2.6	-1.6±0.8	70.6±1.3	-16.7±1.6	64.4±2.3	-14.5±3.2
	GIN	76.7±2.6	-4.0±3.6	65.2±3.9	+0.0±2.0	51.6±2.6	-0.6±1.3	65.3 ±2.9	-17.0±2.3	61.4±3.8	-20.9±2.9
ERGN. [131]	GCN	63.5±2.4	-42.3±0.7	54.2±3.9	-30.3±1.9	52.4±3.3	+0.6±1.4	63.3±1.7	-18.1±0.9	60.7±2.8	-26.6±3.3
	GAT	71.1±2.5	-34.3±1.0	65.5±0.3	-20.4±3.9	51.4±2.2	-7.2±3.2	63.5±2.4	-19.5±1.9	61.3±1.7	-25.1±0.8
	GIN	68.3±0.4	-35.4±0.4	57.7±3.1	-36.4±1.3	42.7±3.9	-13.0±2.1	69.2±1.8	-11.8±1.4	61.8±4.7	-23.4±7.9
TWP [57]	GCN	68.9±0.9	-5.7±1.5	60.5±3.8	-0.3±4.4	50.6±2.0	-4.8±1.1	75.6±0.3	-10.4±0.5	69.9±0.4	-9.0±1.1
	GAT	81.3±3.2	-14.4±1.5	69.8±1.5	-8.9±2.6	54.0±1.8	-2.1±1.9	75.8±0.5	-5.9±0.3	69.3±2.3	-8.9±1.5
	GIN	73.7±3.2	-3.9 ±2.6	68.9±0.7	-2.4±1.9	49.9±1.9	-3.6±2.0	76.6±1.8	-11.3±1.1	69.9±1.4	-10.3±2.7
Join.	GCN	93.7± 0.5	-	78.9±0.4	-	57.0±0.9	-	77.2±0.8	-	72.9±1.2	-
	GAT	93.9± 0.9	-	79.3±0.8	-	57.1±0.9	-	81.8±0.3	-	73.7±2.4	-
	GIN	93.2± 1.2	-	78.7±0.9	-	56.9±0.6	-	82.3±1.9	-	77.9±2.1	-
HPNs		**93.7±1.5**	**+0.6±1.0**	**79.0±0.9**	**-0.6±0.7**	**56.8±1.4**	**-0.9±0.9**	**85.8± 0.7**	**+0.6±0.9**	**80.1±0.8**	**+2.9±1.0**

only 1 example. Similar to OGB-Arxiv, we reorder the classes and the class indices of each tasks are: {(4, 7), (6, 3), (12, 2), (0, 8), (1, 13), (16, 21), (9, 10), (18, 24), (17, 5), (11, 42), (15, 20), (19, 23), (14, 25), (28, 29), (43, 22), (36, 44), (26, 37), (32, 31), (30, 27), (34, 38), (41, 35), (39, 33), (45, 40)}.

4.3.2 Experimental Setup and Evaluation Metrics

To perform continual graph representation learning with new categories of nodes continuously emerging, like the traditional continual learning works, we split each dataset into multiple tasks and train the model on them sequentially. Each new task brings a subgraph with new categories of nodes and associated edges, *e.g.*, task 1 contains classes 1 and 2, task 2 contains new classes 3 and 4, *etc.* Each model is trained on a sequence of tasks, and the performance will be evaluated on all previous tasks. Specifically, we adopt accuracy mean (AM) and

forgetting mean (FM) as metrics for evaluation. After learning on all tasks, the AM and FM are computed as the average accuracy and the average accuracy decrease on all previous tasks. Given a sequence of T tasks, we denote the model's accuracy on the q-th task after learning the p-th task as $\mathbf{M}_{p,q}$, then $\{\mathbf{M}_{T,q}|q = 1, .., T\}$ are the model' accuracy on each learnt task after learning the entire sequence. Accordingly, the mathematical formulations of AM and FM are:

$$\text{AM} = \frac{\sum_{q=1}^{T} \mathbf{M}_{T,q}}{T}, \tag{4.17}$$

$$\text{FM} = \frac{\sum_{q=1}^{T-1} (\mathbf{M}_{T,q} - \mathbf{M}_{q,q})}{T-1}. \tag{4.18}$$

Negative FM indicates the existence of forgetting, zero FM denotes no forgetting and positive FM denotes positive knowledge transfer between tasks. For HPNs, we set $d_a = d_n = d_c = 16$, $l_a = l_r = 22$, and $h = 2$. The threshold t_A, t_N, and t_C are selected by cross validation on $\{0.01, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4\}$. The experiments on the important hyperparameters are provided in Section 4.3.6. All experiments are repeated 5 times with Nvidia Titan Xp GPU. The average performance and standard deviations are reported.

4.3.3 Comparisons with Baseline Methods

We compare HPNs with various baseline methods. Experience Replay based GNN (ERGNN) [129] and Topology-aware Weight Preserving (TWP) [57] are developed for continual graph representation learning. The others approaches, including Elastic Weight Consolidation (EWC) [44], Learning without Forgetting (LwF) [52], Gradient Episodic Memory (GEM) [59], and Memory Aware Synapses (MAS) [3]) are popular continual learning methods for Euclidean data. All the baselines are implemented based on three popular backbone models, *i.e.*, Graph Convolutional Networks (GCNs) [43], Graph Attentional Networks (GATs) [87], and Graph Isomorphism Network (GIN) [105].

Note that Joint training (Join.) in Table 4.2 does not represent continual learning. It allows a model to access data of all tasks at any time and thus is often used as an upper bound for continual learning.[88]. Therefore, FM is not applicable to Joint training approach.

	GAT			HPNs		
Te. \ Tr.	$\mathcal{T}_1$	$\mathcal{T}_{1,2}$	$\mathcal{T}_{1,2,3}$	$\mathcal{T}_1$	$\mathcal{T}_{1,2}$	$\mathcal{T}_{1,2,3}$
$\mathcal{T}_1$	94.12%	47.96%	71.49%	95.02%	95.93%	94.57%
$\mathcal{T}_2$		93.30%	49.68%		92.66%	92.22%
$\mathcal{T}_3$			94.44%			96.83%

TABLE 4.3. Accuracy (%) changes of GAT and HPNs.

	GAT+GEM			GAT+MAS		
Te. \ Tr.	$\mathcal{T}_1$	$\mathcal{T}_{1,2}$	$\mathcal{T}_{1,2,3}$	$\mathcal{T}_1$	$\mathcal{T}_{1,2}$	$\mathcal{T}_{1,2,3}$
$\mathcal{T}_1$	93.21%	91.40%	66.97%	94.12%	90.50%	90.05%
$\mathcal{T}_2$		81.21%	81.43%		77.97%	70.84%
$\mathcal{T}_3$			63.89%			93.25%

TABLE 4.4. Accuracy (%) changes of GAT+GEM and GAT+MAS.

In Table 4.2, we observe that regularization based approaches, *e.g.*, EWC and TWP, generally obtain lower forgetting, but the accuracy (AM) is limited by the constraints. However, the forgetting problem of regularization based methods will become increasingly severe when the number of tasks is relatively large, as shown in Section 4.3.5. Memory replay based methods such as GEM achieve better performance without using any constraint. However, the memory consumption is higher (Section 4.3.7). HPNs significantly outperform all baselines without inheriting their limitations. Compared to regularization based methods, HPNs do not impose constraints to limit the model's expressiveness, therefore the performance is much better. Compared to memory replay based methods, HPNs do not only perform better but also are memory efficient as shown in Section 4.3.7. Joint training (Join.) achieves comparable performance to HPNs on small datasets but is significantly worse on large OGB datasets. This is because joint training (Join.) is a multi-task setting, inter-task interference may cause negative transfer, which is not obvious on small datasets with only a few tasks but becomes prominent on large datasets with tens of tasks. In HPNs, different tasks can choose different combinations of the parameters and thus task interference is dramatically alleviated.

TABLE 4.5. Ablation study on prototypes of different levels of prototypes over Cora.

Conf.	A-p.	N-p.	C-p.	AM%	FM%
1	✓			89.2±1.3	-0.1±0.5
2	✓	✓		91.7±1.1	-0.2±0.8
3	✓	✓	✓	93.7±1.5	+0.6±1.0

TABLE 4.6. Ablation study on different loss terms over Cora.

Conf.	$\mathcal{L}_{cls}$	$\mathcal{L}_{div}$	$\mathcal{L}_{dis}$	AM%	FM%
1	✓			92.4±1.3	+0.8±0.7
2	✓	✓		92.9±1.1	+0.3±1.0
3	✓		✓	92.8±0.9	+0.0±1.2
4	✓	✓	✓	93.7±1.5	+0.6±1.0

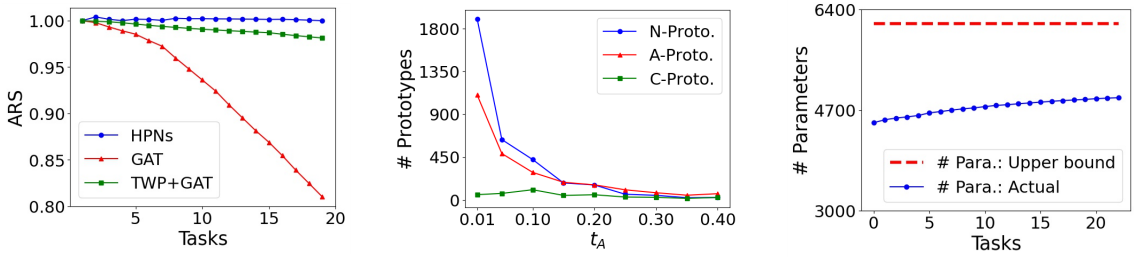


FIGURE 4.2. Left: dynamics of ARS for continual learning tasks on OGB-Arxiv. Middle: impact of t_A on the number of prototypes in HPNs over Cora. Right: dynamics of memory consumption of HPNs on OGB-Products.

Among all the models in Table 4.2, we could observe several kinds of forgetting behavior among the baselines. First, some of them are with low AM and severe forgetting (negative FM with large |FM|) like the pure GNNs in the first 3 rows of Table 4.2. These models may perform well on individual tasks, but the AM is brought down by catastrophic forgetting. To explain this in details, we expand the results of GAT without any continual learning techniques in Table 4.3, in which Tr. denotes on which tasks has the model been trained and Te. denotes on which task the model is tested.

The first row of Table 4.3 shows the model performance on task 1 after it has been trained on the task 1, on the first two tasks, and on the first three tasks. We see that GAT performs well on each individual task it has just learnt. But after being trained on more tasks, the

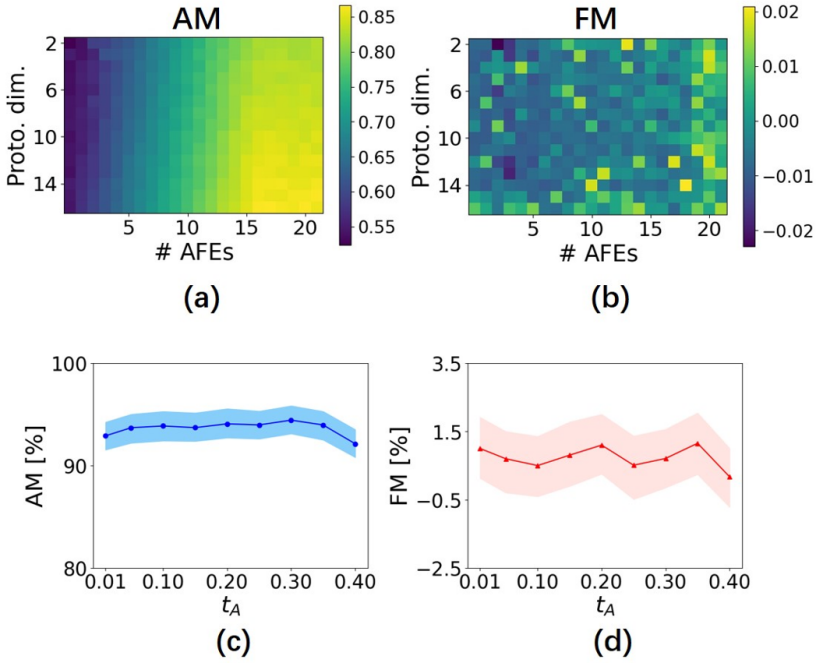


FIGURE 4.3. (a) and (b) are AM and FM of HPNs with different number of AFEs and prototype dimensions on OGB-Arxiv. (c) and (d) are AM and FM change with when t_A varies on Cora.

performance on previous tasks drops dramatically, making the AM to be relatively low. In contrast, the performance change on previous tasks of HPNs is also shown in Table 4.3, and we could see that HPNs can well maintain the performance on previous tasks throughout the training on all tasks.

Second, there are also some models without severe forgetting but the still low AM. Typically these are the models which preserve the performance on previous tasks with certain constraints on the models. These constraints can indeed alleviates forgetting, but it also limits the model's flexibility in learning new tasks, and thus the performance on new tasks will degrade. For example, comparing Table 4.3 with Table 4.4, GAT achieves accuracy higher than 93% on individual tasks, but for GAT+MAS, although the forgetting on task 1 is alleviated, the performance on task 2 drops significantly. GAT+GEM also suffers from the similar problem.

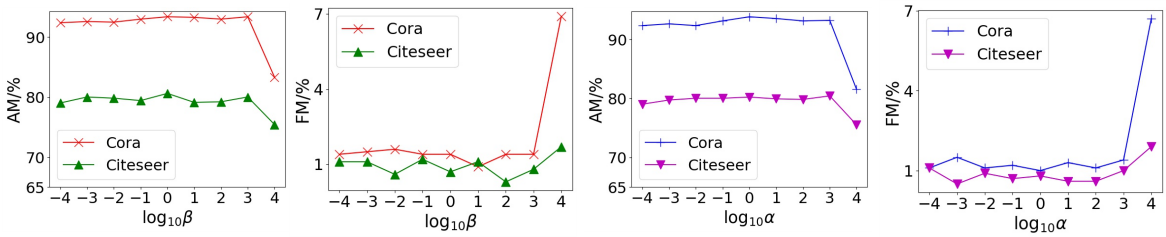


FIGURE 4.4. The left two figures show the results of fixing $\alpha = 1.0$ and tuning β from 0.0001 to 1000. The right two figures show the results of fixing $\beta = 1.0$ and tuning α. Logarithmic horizontal axis is adopted, and the results are obtained on both Cora and Citeseer datasets.

4.3.4 Ablation Study

We conduct ablation studies on different levels of prototypes and different combinations of three loss terms. In Table 4.5, we show the performance of HPNs when A-, N-, and C-Prototypes are gradually added (Cora dataset). We notice both AM and FM of HPNs increase when higher level prototypes are considered. This suggests that high level prototypes can enhance the model's performance and robustness against forgetting. The effect of different combinations of loss terms are shown in Table 4.6. The first three rows show that adding $\mathcal{L}_{div}$ or $\mathcal{L}_{dis}$ with $\mathcal{L}_{cls}$ may slightly improve the performance. By jointly considering these three terms, the performance (AM) can be further improved. This is because $\mathcal{L}_{div}$ pushes different AFEs away from each other and $\mathcal{L}_{dis}$ makes the prototypes of each AFE be more close to its output. Jointly considering $\mathcal{L}_{div}$ and $\mathcal{L}_{dis}$ with $\mathcal{L}_{cls}$ can make the prototype space better separated as shown in Section 4.3.8.

4.3.5 Learning Dynamics

For continual learning, it is important to memorize previous tasks after learning each new task. To measure this, instead of directly measuring the average accuracy on previous tasks which may mix up the accuracy change caused by forgetting and task differences, we develop a new metric, *i.e.*, average retaining score (ARS), to address this problem. Specifically, after learning on a task $\mathcal{T}^i$, the ratio between the model's accuracy on a previous task $\mathcal{T}^{i-m}$ and

TABLE 4.7. Final parameter amount for models trained on OGB-Products

	None	EWC	LwF	GEM	MAS	ERGNN	TWP	Joint	**HPNs**
GCN	2,336	46,720	4,672	2,202,336	2,336	6,738	9,344	2,336	4,908
GAT	20,032	400,640	40,064	2,220,032	20,032	24,432	80,128	20,032	
GIN	2,352	47,040	4,704	2,202,352	2,352	6,752	9,408	2,352	

its accuracy on $\mathcal{T}^{i-m}$ after it had been just learned on $\mathcal{T}^{i-m}$ is defined as the retaining ratio. Then the ARS is the average retaining ratio of all previous tasks after learning a new task.

Figure 4.2(left) shows the ARS change of HPNs and two baselines. GAT represents the models without continual learning techniques. TWP+GAT is the best baseline in terms of forgetting. GAT forgets quickly, while TWP significantly alleviates the forgetting problem for GAT. But as more tasks come in, the forgetting of TWP+GAT increases. As different tasks require different parameters, TWP+GAT (regularization based) is seeking a trade-off between old and new tasks. With more new tasks, TWP+GAT tends to gradually adapt to new tasks and forget old ones. On contrary, HPNs maintain the ARS very well. This is because HPNs learn prototypes to denote the common basic features and learning new tasks does not hurt the parameters for old tasks. New tasks can be handled with new combinations of the existing basic prototypes. If necessary, new prototypes can be established for more expressiveness.

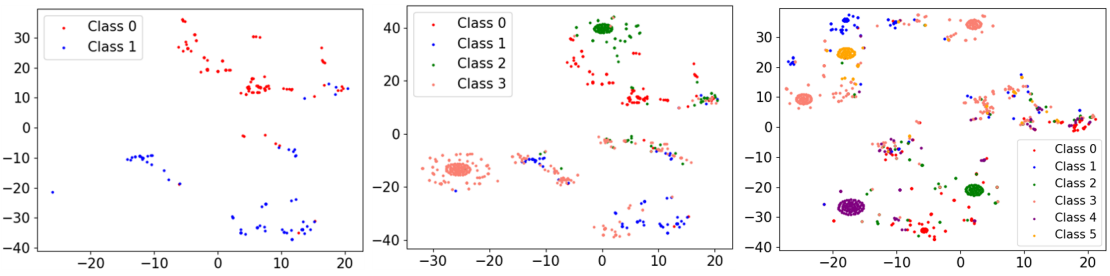


FIGURE 4.5. Visualization of hierarchical prototype representations of nodes in the test set of Cora.

4.3.6 Parameter Sensitivity

As discussed in Section 4.2.5, the number of AFEs and the prototype dimensions are key factors in determining the continual learning capability and memory consumption. Here, we conduct experiments with different number of AFEs and prototype dimensions to justify the

theoretical results. We keep the dimensions of different prototypes equal and the number of two types of AFEs equal for simplicity.

As shown in Figure 4.3(a) and (b), larger dimensions and the number of AFEs yield better AM and FM, which is consistent with Theorem 2. Besides, AM is mostly determined by the number of AFEs since HPNs compose prototypes with different AFEs to represent each target node. The number of possible combinations determines its expressiveness. Considering the above results and the bound (Theorem 1) for the number of prototypes, using large number of AFEs and small dimension can ensure both high performance and low memory usage, as verified in Section 4.3.7.

We also evaluate the effectiveness of HPNs when prototype thresholds vary from 0.01 to 0.4. In Figure 4.3(c) and (d), we observe that the performance (AM and FM) of HPNs are generally stable when t_A varies and slightly better when t_A is between 0.2 and 0.3. This is because when t_A is too small or too large, we will have too many or too less prototypes (consistent with Theorem 1) as shown in Figure 4.2(middle), which may cause the problem of overfitting or underfitting.

Finally, we explore the influence of the hyperparameters of the auxiliary losses (α and β in Equation 4.13) on the performance. As shown in Figure 4.4, we fix $\alpha = 1.0$ or $\beta = 1.0$ in turn, and tune the other one to check the parameter sensitivity.

From Figure 4.4, we can observe that the performance is relatively robust when α and β vary from 0.0001 to 1000. Through monitoring the training process, we suggest the reason is that $\mathcal{L}\_div$ and $\mathcal{L}\_dis$ decrease faster than $\mathcal{L}\_cls$, thus the total loss is not very sensitive to the coefficients on them. Based on this observation, we choose to simply set $\alpha = \beta = 1$ in implementations.

In this subsection, we systematically studied the dependency of the model on the hyperparameters. According to the results, HPNs are robust against most of the hyperparameters. The number of AFEs will cause significant difference but the relation between the performance and the number of AFEs is simple. For a given dataset, the performance first increases with

the number of AFEs, and becomes stable when the number AFEs reaches a certain threshold (*i.e.* large enough).

4.3.7 Memory Consumption

We compare memory consumption of different methods, as well as an explicitly theoretical memory upper bound, with the baselines on OGB-Products (the largest dataset). We also show the actual memory consumption of HPNs in the process of continual learning. We measure the total number of parameters in our model including not only the prototypes but also the parameters of the other modules like the AFEs. The number of prototypes is converted into the number of parameters. For example, an 8-dimensional prototype is counted as 8 parameters, since the prototypes and the other parameters use the same data type with the same precision. Similarly, for other models in Table 4.7, we also show the total number of parameters. For the constant memory models like EWC, LwF, etc., only the trainable parameters are counted. For the memory based methods like GEM, the space consumption of the memory buffer is also counted.

In Table 4.7, even on the dataset with millions of nodes and 23 tasks, HPNs can accommodate all tasks with a small amount of parameters. Besides, the dynamic change of parameter amount is shown in Figure 4.2(right). The red dashed line denotes the theoretical upper bound (6,163). In Figure 4.2(right), we notice the actual memory usage of HPNs is much lower than the upper bound. Moreover, even the upper bound is among the lowest for memory consumption compared to baselines. The model we use here is the same as the one in Section 4.3.3

4.3.8 Visualization

To concretely show the prototype representations generated by HPNs, we apply t-SNE [62] to visualize the node representations of the Cora dataset (test set) after learning each task, as shown in Figure 4.5. The three figures display the representations generated after each of the three tasks arrives sequentially. Each task contains two classes denoted with different colors,

i.e. (red, blue), (green, salmon), and (purple, orange). In Figure 4.5, with new tasks coming in sequentially from left to right, the representations are consistently well separated, which is beneficial for downstream tasks.

4.4 Proof Details

4.4.1 Overview

In this section, we provide detailed explanations and proofs for the theoretical results.

4.4.2 Memory consumption upper bound

The proof and detailed analysis on the memory consumption upper bound has already been given earlier. In this subsection, we give the computation of a specific case of the general theoretical results.

Although the general formulation of the upper bound is not available, we can specially compute $\max_N S(d_a, N, 1 - t_A)$ for certain ns, and verify it with experiments. For example, when $n = 2$, the distribution becomes distributing points on a circle with unit radius. Then, $\max_N S(d_a, N, 1 - t_A)$ can be obtained by evenly distributing the points on the circle with an interval of t_A. Finally, the explicit value of $\max_N S(d_a, N, 1 - t_A)$ can be formulated as:

$$\max_N S(d_n, N, 1 - t_N) = \frac{2\pi}{\arccos(1 - t_A)}, \tag{4.19}$$

then we have:

$$n_A \leqslant (l_a + l_r)\frac{2\pi}{\arccos(1 - t_A)}. \tag{4.20}$$

And the upper bound of the number of N- and C-prototypes can be formulated similarly.

4.4.3 Task distance preserving

In this subsection, we give proofs and detailed analysis on the Theorem 2. At below, Lemma 2, Lemma 3, Lemma 4, and Corollary 1 are from existing knowledge ranging from geometry to linear algebra. The other parts are of our own contributions.

In continual learning, the key challenge is to overcome the catastrophic forgetting, which refers to the performance degradation on previous tasks after training the model on new tasks. Based on our model design, we formulate this as: whether learning new tasks affects the representations the model generates for old task data. First, we give definitions on the tasks and task distances:

DEFINITION 2 (Task set). *The p-th task in a sequence is denoted as $\mathcal{T}^p$ and contains a subgraph $\mathcal{G}_p$ consisting of nodes belonging to some new categories. We denote the associated node set and adjacency matrix as $\mathbb{V}_p$ and A_p. Each $v_p^i \in \mathbb{V}_p$ has a feature vector $\mathbf{x}(v_p^i)$ and a label $y(v_p^i)$.*

Then, the reason for catastrophic forgetting is that different tasks in a sequence are drawn from heterogeneous distributions, making the model sequentially trained on different tasks unable to maintain satisfying performances on previous tasks. Therefore, given the definition of the tasks (Definition 2), we then give a formal definition to quantify the difference between two tasks.

DEFINITION 3 (Task distance). *We define the distance between two tasks as the set distance between the node sets of these two tasks, i.e.*

$$\mathbf{dist}(\mathbb{V}_p, \mathbb{V}_q) = \inf\|\mathbf{x}(v_p^i) - \mathbf{x}(v_q^j)\|, \forall v_p^i \in \mathbb{V}_p, v_q^j \in \mathbb{V}_q.$$

LEMMA 1. *The distance between any two tasks is non-negative, i.e. $\forall i, j \in \{1, ..., \mathrm{M}^T\}, \mathbf{dist}(\mathbb{V}_p, \mathbb{V}_q) \geqslant 0$, where M^T is the number of tasks contained in the sequence.*

The real-world data could be complex and sometimes may even contain noises that are impossible for any model to learn, which needs extra considerations when justifying the effectiveness of the model. Formally, we give the definition of the contradictory data.

DEFINITION 4 (Contradictory data). *$\forall v_p^i \in \mathbb{V}_p, p = 1, ..., \mathrm{M}^T$, if $\exists v_q^j \in \mathbb{V}_q, j = 1, ..., \mathrm{M}^T$, $\mathrm{st.} \forall l \in \mathbb{N}^*, \forall u \in \mathcal{N}^l(v_p^i)$ and $\forall v \in \mathcal{N}^l(v_q^j)$, $\mathbf{x}(u) = \mathbf{x}(v)$ but $y(v_p^i) \neq y(v_q^j)$, then we say $(v_p^i, y(v_p^i))$ and $(v_q^j, y(v_q^j))$ are contradictory data, as it is contradictory for any model to give different predictions for one node based on the same node features and graph structures. ($\mathbb{N}^*$ denotes the set of non-negative integers)*

REMARK 1. *Contradictory data is ignored or simply regarded as outliers in previous works, but in this work, we explicitly analyze its affect for the comprehensiveness of our theory. contradictory data has different situations. If v_p^i and v_q^j are from different tasks, then $y(v_p^i) \neq y(v_q^j)$ is plausible. Because they may be describing a same thing from different aspects. For example, an article from the citation network may be both categorized as 'physics related' and 'computer science related'. In this situation, it would be easy to add an task indicator to the feature of the node, then the feature of v_p^i and v_q^j are no longer equal and are not contradictory data anymore.*

But within one task, contradictory data are most likely to be wrongly labeled, e.g. it does not make sense if an article is both 'related to physics' and 'not related to physics'.

Besides the distance between tasks, the distance between the embeddings obtained by the AFEs will also be a crucial concept in the proof.

DEFINITION 5 (Embedding distance). *Each input node v_p^i is given a set of atomic embeddings $\mathbb{E}_A(v_p^i) = \mathbb{E}_A^{\mathrm{node}}(v_p^i) \cup \mathbb{E}_A^{\mathrm{struct}}(v_p^i)$, where $\mathbb{E}_A^{\mathrm{node}}(v_p^i) = \{\mathbf{a}_i^j | j \in \{1, ..., l_a\}\}_p$ containing the atomic node embeddings of v_p^i and $\mathbb{E}_A^{\mathrm{struct}}(v_p^i) = \{\mathbf{r}_i^j | k \in \{1, ..., l_r\}\}_p$ containing the atomic structure embeddings. $\mathbf{a}_i^j \in \mathbb{R}^{d_a}$ and $\mathbf{r}_i^j \in \mathbb{R}^{d_r}$. To define the distance between representations of two nodes, we concatenate the atomic embeddings of each node into a single vector in a higher dimensional space, i.e. each node v_p^i corresponds to a latent vector $\mathbf{z}_p^i = [\mathbf{a}_i^1; ...; \mathbf{a}_i^{l_a}; \mathbf{r}_i^1; ...; \mathbf{r}_i^{l_r}] \in \mathbb{R}^{l_a \times d_a + l_r \times d_r}$. Then we define the distance between representations of two nodes v_p^i and v_q^j as the Euclidean distance between their corresponding latent vector $\mathbf{z}_\mathbf{p}^\mathbf{i}$ and $\mathbf{z}_\mathbf{q}^\mathbf{j}$, i.e. $\mathrm{dist}(\mathbf{z}_p^i, \mathbf{z}_q^j) = \|\mathbf{z}_p^i - \mathbf{z}_q^j\|_2$.*

Then we will give some explanations on the linear algebra related theories.

LEMMA 2 (Bounds for real quadratic forms). *Given a real symmetric matrix* $\mathbf{A}$*, and an arbitrary real vector variable* $\mathbf{x}$*, we have*

$$\lambda_{\min} \leqslant \frac{\mathbf{x}^T\mathbf{A}\mathbf{x}}{\mathbf{x}^T\mathbf{x}} \leqslant \lambda_{\max}, \tag{4.21}$$

where $\lambda_{\min}$ *and* $\lambda_{\max}$ *are the minimum and maximum eigenvalues of matrix* $\mathbf{A}$*.*

LEMMA 3 (Real symmetric matrix). *For a matrix* $\mathbf{A} \in \mathbb{R}^{m\times n}$*,* $\mathbf{A}^T\mathbf{A} \in \mathbb{R}^{n\times n}$ *is a real symmetric matrix,* $rank(\mathbf{A}^T\mathbf{A}) = rank(\mathbf{A})$*, and the non-zero eigenvalues of* $\mathbf{A}^T\mathbf{A}$ *are squares of the non-zero singular values of* $\mathbf{A}$*.*

LEMMA 4 (Rank and number of non-zero singular values). *For a matrix* $\mathbf{A} \in \mathbb{R}^{m\times n}$*, the number of non-zero singular values equals the rank of* $\mathbf{A}$*, i.e.* $rank(\mathbf{A})$

COROLLARY 1. *For a matrix* $\mathbf{A} \in \mathbb{R}^{m\times n}$*. Without loss of generality, we assume* $n \leqslant m$*. If* $\mathbf{A}$ *is column full rank, i.e.* $rank(\mathbf{A}) = n$*, then* $\mathbf{A}$ *has* n *non-zero singular values. Besides,* $rank(\mathbf{A}^T\mathbf{A}) = n$*, and* $\mathbf{A}^T\mathbf{A}$ *has* n *non-zero singular values.*

Given the explanations above, we then derive the bound for the change of the distance among data, which will be further used for analyzing the separation of data from different tasks.

LEMMA 5 (Embedding distance bound). *Given two nodes* $v_p^i \in \mathbb{V}_p$ *and* $v_q^j \in \mathbb{V}_q$ *with vertex feature* $\mathbf{x}(v_p^i), \mathbf{x}(v_q^j) \in \mathbb{R}^{d_v}$*, their multi-hop neighboring node sets are denoted as* $\bigcup_{l\in\mathbb{N}^*} \mathcal{N}^l(v_p^i)$ *and* $\bigcup_{l\in\mathbb{N}^*} \mathcal{N}^l(v_q^j)$*. The AFEs for generating atomic embeddings are* $\mathrm{AFE}_{\mathrm{node}} = \{\mathbf{A}_i \in \mathbb{R}^{d_a\times d_v} | i \in \{1, ..., l_a\}$ *and* $\mathrm{AFE}_{\mathrm{struct}} = \{\mathbf{R}_j \in \mathbb{R}^{d_r\times d_v} | j \in \{1, ..., l_r\}$*, corresponding to matrices for atomic node embeddings and atomic structure embeddings, respectively. Then, the square distance*

$$\mathrm{dist}^2(\mathbf{z}_p^i - \mathbf{z}_q^j) = \|\mathbf{z}_p^i - \mathbf{z}_q^j\|_2^2 \geqslant \lambda_{\min}(\|\mathbf{x}(v_p^i) - \mathbf{x}(v_q^j)\|_2^2 + \textstyle\sum_{k=1}^{l_r} \|\mathbf{x}(u_k) - \mathbf{x}(\nu_k)\|_2^2),$$

if $l_a \times d_a + l_r \times d_r \geqslant d_v$*, where* u_k *are nodes sampled from* $\bigcup_{l\in\mathbb{N}^*} \mathcal{N}^l(v_p^i)$*,* ν_k *are nodes sampled from* $\bigcup_{l\in\mathbb{N}^*} \mathcal{N}^l(v_q^j)$*,* λ_i *are the eigenvalues of* $\mathbf{W}^T\mathbf{W}$*, and* $\mathbf{W} \in \mathbb{R}^{(l_r+1)d_v\times(l_a d_a + l_r d_r)}$ *is constructed with the matrices in* $\mathrm{AFE}_{\mathrm{node}}$ *and* $\mathrm{AFE}_{\mathrm{struct}}$*. Specifically,* $\mathbf{W}$ *is a block matrix constructed as follows:*

1. $\mathbf{W}_{1:l_a d_a, 1:d_v}$ *are filled by the concatenation of* $\{\mathbf{A}_i | i = 1, ..., l_a\}$*, i.e.* $[\mathbf{A}_1; ...; \mathbf{A}_{l_a}] \in \mathbb{R}^{l_a d_a \times d_v}$.

2. For $\mathbf{W}_{l_a d_a+1:l_a d_a+l_r d_r, 1:(l_r+1)d_v}$*, the construction is first filling* $\mathbf{W}_{l_a d_a+(k-1)d_r:l_a d_a+kd_r, kd_v:(k+1)d_v}$ *with* $\mathbf{R}_k$*,* $k = 1, ..., l_r$.

3. For other parts, fill with zeros.

PROOF. Given vertex v_p^i, we concatenate its feature vector with the l_r neighbors sampled from $\bigcup_{l \in \mathbb{N}^*} \mathcal{N}^l(v_p^i)$, i.e. $\mathbf{x}'_{p,i} = [\mathbf{x}(v_p^i); \mathbf{x}(u_1); ...; \mathbf{x}(u_{l_r})] \in \mathbb{R}^{(l_r+1)d \times 1}, u_j \in \bigcup_{l \in \mathbb{N}^*} \mathcal{N}^l(v_p^i)$. Then with the constructed block matrix $\mathbf{W}$, we could formulate the generation of $\mathbf{z}_p^i$ as: $\mathbf{z}_p^i = \mathbf{W}\mathbf{x}'_{p,i}$.

Similarly, we can formulate $\mathbf{z}_q^j$ for another vertex v_q^j. And their distance can be formulated as:

$$\mathrm{dist}(\mathbf{z}_p^i, \mathbf{z}_q^j) = \|\mathbf{z}_p^i - \mathbf{z}_q^j\|_2 = \sqrt{(\mathbf{z}_p^i - \mathbf{z}_q^j)^T(\mathbf{z}_p^i - \mathbf{z}_q^j)},$$

where $\mathbf{z}_p^i - \mathbf{z}_q^j)^T(\mathbf{z}_p^i - \mathbf{z}_q^j)$ can be further expanded as:

$$\begin{aligned}(\mathbf{z}_p^i - \mathbf{z}_q^j)^T(\mathbf{z}_p^i - \mathbf{z}_q^j) &= (\mathbf{W}\mathbf{x}'_{p,i} - \mathbf{W}\mathbf{x}'_{q,j})^T(\mathbf{W}\mathbf{x}'_{p,i} - \mathbf{W}\mathbf{x}'_{q,j}) \\ &= \big(\mathbf{W}(\mathbf{x}'_{p,i} - \mathbf{x}'_{q,j})\big)^T\big(\mathbf{W}(\mathbf{x}'_{p,i} - \mathbf{x}'_{q,j})\big) \\ &= (\mathbf{x}'_{p,i} - \mathbf{x}'_{q,j})^T\mathbf{W}^T\mathbf{W}(\mathbf{x}'_{p,i} - \mathbf{x}'_{q,j})\end{aligned}$$

According to lemma 3, $\mathbf{W}^T\mathbf{W}$ is a real symmetric matrix, with lemma 2, we have:

$$\frac{(\mathbf{x}'_{p,i} - \mathbf{x}'_{q,j})^T\mathbf{W}^T\mathbf{W}(\mathbf{x}'_{p,i} - \mathbf{x}'_{q,j})}{(\mathbf{x}'_{p,i} - \mathbf{x}'_{q,j})^T(\mathbf{x}'_{p,i} - \mathbf{x}'_{q,j})} \geqslant \lambda_{min}$$

According to lemma 3, with $l_a d_a + l_r d_r \geqslant (l_r + 1)d_v$ and the constraint of column full rank on $\mathbf{W}$, $\mathbf{W}^T\mathbf{W} \in \mathbb{R}^{(l_r+1)\times(l_r+1)}$ has $l_r + 1$ positive eigenvalues, thus $\lambda_{min} > 0$. Then we

decompose $(\mathbf{x}'_{p,i} - \mathbf{x}'_{q,j})^T(\mathbf{x}'_{p,i} - \mathbf{x}'_{q,j})$ as:

$$
\begin{aligned}
&(\mathbf{x}'_{p,i} - \mathbf{x}'_{q,j})^T(\mathbf{x}'_{p,i} - \mathbf{x}'_{q,j}) \\
=& \sum_{k=0}^{(l_r+1)d-1} \left((\mathbf{x}'_{p,i})_k - (\mathbf{x}'_{q,j})\right)^2 \\
=& \sum_{k=1}^{(l_r+1)d_v} \left((\mathbf{x}'_{p,i})_k - (\mathbf{x}'_{q,j})\right)^2 \\
=& \sum_{k=1}^{d_v} \left((\mathbf{x}'_{p,i})_k - (\mathbf{x}'_{q,j})_k\right)^2 + \sum_{k=1}^{l_r} \sum_{m=kd_v+1}^{(k+1)d_v} \left((\mathbf{x}'_{p,i})_k - (\mathbf{x}'_{q,j})_k\right)^2 \\
=& \|\mathbf{x}(v_p^i) - \mathbf{x}(v_q^j)\|_2^2 + \sum_{m=1}^{l_r} \|\mathbf{x}(u_m) - \mathbf{x}(\nu_k)\|_2^2
\end{aligned}
$$

$$\therefore \operatorname{dist}^2(\mathbf{z}_p^i - \mathbf{z}_q^j) = \|\mathbf{z}_p^i - \mathbf{z}_q^j\|_2^2 \geqslant \lambda_{\min}(\|\mathbf{x}(v_p^i) - \mathbf{x}(v_q^j)\|_2^2 + \textstyle\sum_{k=1}^{l_r} \|\mathbf{x}(u_k) - \mathbf{x}(\nu_k)\|_2^2)$$

□

The key point in these theories is that for any task sequence with certain distance among the tasks, there exists a configuration that ensures HPNs to be capable of preserving the task distance after projecting the data into the hidden space, so that only the prototypes associated with the current task are refined and the prototypes corresponding to the other tasks are preserved. Specifically, theorem on zero-forgetting can be formulated as follows:

THEOREM 3 (Task distance preserving). *For* HPNs *trained on consecutive tasks $\mathcal{T}^p$ and $\mathcal{T}^{p+1}$. If $l_a d_a + l_r d_r \geqslant (l_r + 1)d_v$ and $\mathbf{W}$ is column full rank, then as long as $t_A < \lambda_{\min}(l_r+1)\mathbf{dist}(\mathbb{V}_p, \mathbb{V}_{p+1})$, learning on $\mathcal{T}^{p+1}$ will not modify representations* HPNs *generate for data from $\mathcal{T}^p$, i.e. catastrophic forgetting is avoided.*

In Theorem 3, λ_i is eigenvalues of the $\mathbf{W}^T\mathbf{W}$, where $\mathbf{W}$ is the matrix mentioned before constructed via AFEs. d_v, d_a and d_r are dimensions of data, atomic node embeddings, and atomic structure embeddings.

PROOF. Following the proofs above, suppose two nodes v_p^i and v_q^j are embedded into $\mathbf{z}_p^i$ and $\mathbf{z}_q^j$ with the embedding module. Then the distance between $\mathbf{z}_p^i$ and $\mathbf{z}_q^j$ could be formulated as:

$$\mathrm{dist}(\mathbf{z}_p^i, \mathbf{z}_q^j) = ||\mathbf{z}_p^i - \mathbf{z}_q^j||_2 = \sqrt{(\mathbf{z}_p^i - \mathbf{z}_q^j)^T(\mathbf{z}_p^i - \mathbf{z}_q^j)}$$

According to lemma 5, we have $\mathrm{dist}^2(\mathbf{z}_p^i, \mathbf{z}_q^j) = ||\mathbf{z}_p^i - \mathbf{z}_q^j||_2^2 \geqslant \lambda_{\min}(||\mathbf{x}(v_p^i) - \mathbf{x}(v_q^j)||_2^2 + \sum_{k=1}^{l_r} ||\mathbf{x}(u_k) - \mathbf{x}(\nu_k)||_2^2)$.

$\because v_p^i \in \mathbb{V}_p, v_q^j \in \mathbb{V}_q$,

$\therefore ||\mathbf{x}(v_p^i) - \mathbf{x}(v_q^j)||_2^2 \geqslant \mathrm{dist}^2(\mathbb{V}_p, \mathbb{V}_q)$.

Similarly, $||\mathbf{x}(u_k) - \mathbf{x}(\nu_k)||_2^2 \geqslant \mathrm{dist}^2(\mathbb{V}_p, \mathbb{V}_q)$, for $\forall k$.

$\therefore ||\mathbf{z}_p^i, \mathbf{z}_q^j||_2^2 \geqslant \lambda_{\min}(l_r + 1)\mathrm{dist}^2(\mathbb{V}_p, \mathbb{V}_q)$

$\therefore \mathrm{dist}(\mathbf{z}_p^i, \mathbf{z}_q^j) = ||\mathbf{z}_p^i - \mathbf{z}_q^j||_2 \geqslant \sqrt{\lambda_{\min}(l_r + 1)\mathrm{dist}^2(\mathbb{V}_p, \mathbb{V}_q)}$

$\therefore$ If $t_A < \sqrt{\lambda_{\min}(l_r + 1)\mathrm{dist}^2(\mathbb{V}_p, \mathbb{V}_q)}$, the embeddings of two nodes from two different tasks will not be assigned to same A-prototypes.

Above all, if the conditions in Theorem 3 are satisfied, learning on new tasks will not modify the prototypes for previous tasks. Besides, the data from previous tasks will be exactly matched to the correct prototypes after training the model on new tasks. In practice, the conditions may not be easy to be satisfied all the time. However, as mentioned before, the bound given in Theorem 3 is not tight, thus fully satisfying the conditions may not be necessary. Therefore, in the experimental section, we practically show how the important factors included in these conditions influence the performance (Section 4.3.6). The results demonstrate that the more we satisfy the conditions, the better performance we will obtain, and certain factors (number of AFEs) influence more than the others. □

REMARK 2. *When* $\mathrm{dist}(\mathbb{V}_p, \mathbb{V}_q) = 0$*, i.e. there exists a non-empty set* $\mathbb{V}_\cap = \mathbb{V}_p \cap \mathbb{V}_q$, st. $\mathrm{dist}(\mathbb{V}_p \setminus \mathbb{V}_\cap, \mathbb{V}_q \setminus \mathbb{V}_\cap) > 0$*, then Theorem 3 holds. As for the* $\mathbb{V}_\cap$ *containing examples exactly same in* $\mathbb{V}_p$ *and* $\mathbb{V}_q$*, there are two situations:*

1. $\forall v \in \mathbb{V}_{\cap}$, $y_p(v) = y_q(v)$, *where* $y_p(\cdot)$ *and* $y_q(\cdot)$ *denote the associated labels in task* p *and* q

2. $\exists v \in \mathbb{V}_{\cap}$, $y_p(v) \neq y_q(v)$

For situation 1, $\mathbb{V}_{\cap}$ *will not cause forgetting on the previous task, as these shared data are exactly same and will optimize the model to same direction. For situation 2, if no task indicator is provided, then these data are contradictory data, if task indicator is provided, then the indicator could be merged into the feature vector of the node, i.e.* $\mathbf{x}(v_p)$*, then* v_p *will not belong to* $\mathbb{V}_{\cap}$*.*

4.5 Conclusion

In this work, we proposed Hierarchical Prototype Networks (HPNs), to continuously extract different levels of abstract knowledge (in the form of prototypes) from streams of tasks on graph representation learning. The continual learning performance of HPNs is both theoretically analyzed and experimentally justified from different perspectives on multiple public datasets with up to millions of nodes and tens of millions of edges. Besides, the memory consumption of HPNs is also theoretically analyzed and experimentally verified.

HPNs provide a general framework for continual graph representation learning and can be further extended to tackle more challenging tasks. In the future, we will extend HPNs to heterogeneous graphs by adapting the AFEs and prototypes to capture multiple types of nodes and edges. Moreover, we will also equip the prototypes with connections to denote the relationship among the abstract knowledge and investigate more challenging graph reasoning tasks.

CHAPTER 5

Sparsified Subgraph Memory for Continual Graph Representation Learning

5.1 Introduction

This chapter introduces a memory-replay based CGL technique. Memory replay, inspired by research on cognitive science [66, 48], has demonstrated state-of-the-art performance in various classical continual learning tasks. The key idea is to replay a subset of data samples from previous tasks over the model while learning the new tasks [74, 59, 5]. Due to its success, memory replay is also adopted for continual learning on graph data by storing and replaying representative nodes [129]. For graph data, however, only storing a subset of nodes for replay will neglect the important topological information. Since the properties of the target node are not only determined by itself but also by its neighbors, in this work, we propose to store computation subgraphs to explicitly preserve the topological information for memory replay. Directly storing computation subgraphs, however, will trigger the memory explosion problem. Supposing the average node degree is d, then in a L-layer GNN following the message passing paradigm [32], the size of the neighboring nodes in the computation subgraph of a node will be $\mathcal{O}(d^L)$, let alone the associated edges which are typically several orders of magnitude more than the nodes. For instance, in the Reddit dataset, the average node degree is 492, and the maximal degree is 21,657. Obviously, directly storing the entire computation subgraphs is infeasible.

To utilize the topological information while maintaining a tractable space complexity at the same time, we propose to sparsify a computation subgraph to a fixed size before storing

it into the memory. Specifically, to sparsify the computation subgraph of a node v while maintaining the connectivity of sparsified subgraphs at the same time, we start by sampling the 1-hop neighbors, and then iteratively sample the higher-order neighbors hop by hop. In this way, since we can fix the number of nodes to sample at each hop, the size of the sparsified computation subgraph will be independent of the original computation subgraph, *i.e.*, the memory space complexity can be reduced from $\mathcal{O}(d^L)$ to $\mathcal{O}(1)$. In this work, we adopted two sparsification strategies using uniform sampling and degree based sampling, which are simple yet empirically effective. Note that we are aware of other existing graph sparsification techniques [41, 64, 15, 17, 50, 111, 60, 128], however, most of them are not applicable for continual graph representation learning as detailed in Chapter 2.

Based on our thorough empirical studies on four public datasets, SSM outperforms the existing state-of-the-art methods by a large margin in the challenging class-IL scenario. Moreover, the performance of SSM is even comparable to joint training which is the performance upper bound for continual learning. To summarize, our contributions are:

(1) We develop the Sparsified Subgraph Memory (SSM) to store the explicit topological information in the form of sparsified computation subgraphs and perform memory replay based continual graph representation learning.
(2) We resolve the memory explosion problem by sparsifying the subgraphs.
(3) Our proposed SSM outperforms the existing state-of-the-art methods by a large margin, especially in the challenging class-IL scenario.

5.2 Methods

In this section, we first introduce the preliminaries that include the basic concepts, notations, and learning settings. Next, we explain how memory replay works in traditional continual learning with independent data examples and why applying memory replay with GNNs on individual graph nodes can result in severe information loss. After that, we explain the memory explosion problem triggered by directly storing the complete topological information. Finally, we introduce our proposed solution for topology sparsification.

5.2.1 Preliminaries

In this work, we focus on the node-level continual graph representation learning, which aims to continuously accommodate the new types (classes) of emerging nodes (new tasks) and their associated edges without interfering with the performance over existing nodes (previous tasks). With this setting, a model is trained on a sequence of tasks (subgraphs): $\mathcal{S} = \{\mathcal{G}_1, \mathcal{G}_2, ..., \mathcal{G}_T\}$. Each $\mathcal{G}_\tau$ contains nodes belonging to a unique set of classes in its node set $\mathbb{V}_\tau$. The associated edge set is $\mathbb{E}_\tau$, in which an edge e_{uv} denotes the existence of an edge connecting node u and v. $\mathbb{E}_\tau$ is often represented as the adjacency matrix $\mathbf{A}_\tau \in \mathbb{R}^{|\mathbb{V}_\tau|\times|\mathbb{V}_\tau|}$, where every non-zero entry corresponds to an edge in $\mathbb{E}_\tau$. $\mathbf{A}_\tau$ can be normalized as $\hat{\mathbf{A}}_\tau = \mathbf{D}_\tau^{-\frac{1}{2}}\mathbf{A}_\tau\mathbf{D}_\tau^{-\frac{1}{2}}$, where $\mathbf{D}_\tau \in \mathbb{R}^{|\mathbb{V}_\tau|\times|\mathbb{V}_\tau|}$ is the degree matrix that contains the degree of each node (number of connected edges) in its diagonal entries. Each node $u \in \mathbb{V}_\tau$ has a feature vector $\mathbf{x}_u \in \mathbb{R}^d$, and a label $\mathbf{y}_u \in \{0,1\}^C$, where C is the number of all possible classes. GNNs generate the representation for a node u based on a computation subgraph denoted as $\mathcal{G}_{\tau,u}^{sub}$, which is a subgraph of $\mathcal{G}_\tau$. In the following, $\mathcal{G}_u^{sub}$ without the graph index will be used for simplicity. Finally, we denote the L-hop neighbors of u as $\mathcal{N}^L(u)$ containing the nodes within a distance of L from u, *i.e.*, for a node $u \in \mathbb{V}_i$:

$$\mathcal{N}^L(u) = \{v \in \mathbb{V}_\tau | \mathrm{d}(u,v) = L\}, \tag{5.1}$$

where $\mathrm{d}(\cdot,\cdot)$ denotes the shortest path distance between two nodes. Specially, we have $\mathcal{N}^0(u) = \{u\}$. The vast majority of GNNs follow the message passing neural network (MPNN) paradigm [32]. In this work, we focus on the MPNNs and refer all GNNs to MPNNs in the following.

5.2.2 Memory Replay on Graphs

In this subsection, we first introduce how memory replay works in traditional continual learning, and then derive the information loss of directly applying memory replay to store individual graph nodes. Finally, we introduce the challenge of storing the topological information of graph data.

Traditional continual learning can be described as training a model $\mathrm{f}(\cdot;\boldsymbol{\theta})$ on a task sequence of length T with the accompanied datasets $\mathbb{D}_\tau = \{(\mathbf{x}_i, \mathbf{y}_i)_{i=1}^{n_\tau}\}$ ($\tau \in \{1, ..., T\}$). The dataset for τ-th task $\mathbb{D}_\tau$ is only available when learning this task, and becomes inaccessible afterward. To alleviate the forgetting problem, memory replay based methods typically maintain a memory buffer $\mathcal{B}$ containing the representative data from the previous tasks, which are replayed to the model when learning new tasks. A straightforward way to utilize $\mathcal{B}$ is through an auxiliary loss:

$$\mathcal{L} = \underbrace{\sum_{\mathbf{x}_i \in \mathbb{D}_\tau} l(f(\mathbf{x}_i;\boldsymbol{\theta}), \mathbf{y}_i)}_{\text{loss of the current task } \mathcal{L}_\tau} + \lambda \underbrace{\sum_{\mathbf{x}_j \in \mathcal{B}} l(f(\mathbf{x}_j;\boldsymbol{\theta}), \mathbf{y}_j)}_{\text{auxiliary loss } \mathcal{L}_{aux}}, \tag{5.2}$$

where the loss function is denoted as $l(\cdot,\cdot)$, and the contribution of auxiliary loss is balanced by λ. Besides directly optimizing an auxiliary loss, there are also other ways to prevent forgetting with the stored data in $\mathcal{B}$. For example, GEM [59] calibrates the gradients of $\mathcal{L}_\tau$ with the gradients of $\mathcal{L}_{aux}$ to prevent increasing the loss on previous tasks ($\mathcal{L}_{aux}$); iCaRL [74] directly uses the representations of the stored data ($\{f(\mathbf{x}_i;\boldsymbol{\theta}), \mathbf{x}_i \in \mathcal{B}\}$) as prototypes to classify new data. For different approaches, we always have to regenerate their representations based on the buffered data. Traditional continual learning deals with independent data samples, and regenerating the representation $f(\mathbf{x}_i;\boldsymbol{\theta})$ only takes $\mathbf{x}_i$ itself as input. Therefore, the memory space complexity for replaying the representation of one independent example (node) is $\mathcal{O}(1)$.

To capture the rich topological information within a graph, the representation of a node not only depends on itself but also depends on its neighboring nodes and the accompanied edges. This will significantly increase the memory space complexity for replaying a graph node. Without loss of generality, we take the Message Passing Neural Networks (MPNNs) as an example. The rule for updating the hidden representation of a node u at layer $l+1$ is:

$$\mathbf{m}_u^{l+1} = \sum_{v \in \mathcal{N}^1(u)} \mathrm{M}_l(\mathbf{h}_u^l, \mathbf{h}_v^l, \mathbf{x}_{u,v}^e; \boldsymbol{\theta}_l^{\mathrm{M}}), \tag{5.3}$$

$$\mathbf{h}_u^{l+1} = \mathrm{U}_l(\mathbf{h}_u^l, \mathbf{m}_u^{l+1}; \boldsymbol{\theta}_l^{\mathrm{U}}), \tag{5.4}$$

where $\mathbf{h}_u^l$, $\mathbf{h}_v^l$ denote the hidden representations at layer l, $\mathbf{x}_{uv}^e$ denotes the possible edge features, the function $\mathrm{M}_l(\cdot,\cdot,\cdot;\boldsymbol{\theta}_l^{\mathrm{M}})$ generates message $\mathbf{m}_u^{l+1}$ from neighboring nodes, and

the function $\mathrm{U}_l(\cdot,\cdot;\boldsymbol{\theta}_l^{\mathrm{U}})$ aggregates and updates the neighborhood information $\mathbf{m}_u^{l+1}$ into the representation $\mathbf{h}_u^l$ of the target node u. Given a L-layer MPNN, we can simplify the representation of a node u as:

$$\mathbf{h}_u^L = \mathrm{MPNN}(\mathcal{G}_u^{sub};\boldsymbol{\Theta}), \tag{5.5}$$

where $\mathrm{MPNN}(\cdot,\cdot;\boldsymbol{\Theta})$ denotes the composition of the message functions and the update functions at all layers. According to the updating rule, in L-layer MPNNs, $\mathcal{G}_u^{sub}$ would contain the L-hop neighbors $\mathcal{N}^L(v)$ and the accompanied edges.

To perform memory replay over graphs, the naive approach which stores individual nodes will lose the important explicit topological information of the computation subgraph $\mathcal{G}_u^{sub}$. A proper way should be to store representative computation subgraphs of existing tasks. However, due to the rich connections among different nodes, the size of different computation subgraphs varies and could be extremely large. Supposing the average node degree of $\mathcal{G}_v^{sub}$ is d, then the expected space complexity of storing its nodes would be $\mathcal{O}(d^L)$ (number of edges not counted yet), which can easily exceed the memory buffer size. For example, the average degree of the Reddit dataset is 492, and the maximal degree is 21,657, which would easily result in intractable memory consumption.

5.2.3 Graph Sparsification via Neighborhood Sampling

To sparsify a given computation subgraph, directly sampling over the nodes may cause the sparsified subgraph to be disconnected, causing problems for messaging passing over the subgraph. To ensure the connectivity, we sample the nodes iteratively from 1-hop neighbors to the outermost neighbors. At each hop, we only sample from the nodes connected to the selected ones. Specifically, given a computation subgraph $\mathcal{G}_u^{sub}$ containing neighbors from 1-L-hop, we set a fixed memory budget K on the nodes to sample from each hop. Denoting the budget for the l-th hop as k_l, we have $\sum_{l=1}^{L} k_l = K$. Then the detailed sampling procedure is described in Algorithm 3.

Algorithm 2 Computation subgraph sparsification

Input: $\mathcal{G}_u^{sub}$, node set $\mathbb{V}^{sub}$, edge set $\mathbb{E}^{sub}$, memory budget $\{k_l | l = 1, ..., L\}$.
Output: Sparsified computation subgraph $\bar{\mathcal{G}}_u^{sub}$
Initialize a node set $\mathbb{V} = \{u\}$, an empty edge set $\mathbb{E}$.
for each $l \leftarrow 1$ **to** L **do**
    Initialize a temporary node set $\mathbb{V}_{temp} = \{u\}$
    Get the union of the 1-hop neighbors of the nodes in $\mathbb{V}_{temp}$ and excludes selected nodes $\mathbb{V}$, *i.e.* $\bigcup_{v \in \mathbb{V}_{temp}} \mathcal{N}^1(v) \backslash \mathbb{V}$.
    Sample k_l nodes $\mathbb{V}_{samp}$ from $\bigcup_{v \in \mathbb{V}_{temp}} \mathcal{N}^1(v) \backslash \mathbb{V}$ according to a certain probability distribution (*e.g. degree based or uniform sampling*).
    $\mathbb{V} = \mathbb{V} \cup \mathbb{V}_{samp}$
    for each $v \in \mathbb{V}_{temp}$ **do**
        for each $w \in \mathbb{V}_{samp}$ **do**
            if $e_{w,v} \in \mathbb{E}^{sub}$ **then**
                $\mathbb{E} = \mathbb{E} \cup \{e_{w,v}\}$ ▷ Store the accompanied edges
            end if
        end for each
    end for each
    $\mathbb{V}_{temp} = \mathbb{V}_{samp}$
end for each
Construct $\bar{\mathcal{G}}_u^{sub}$ with $\mathbb{V}$ and $\mathbb{E}$.

Since the memory budget K is constant regardless of the graphs or models, the memory space complexity for replaying one node is reduced to $\mathcal{O}(1)$, which is manageable and similar to the traditional continual learning setting. With the sparisified computation subgraphs stored in the Subgraph Episodic Memory $\mathcal{SEM}$, the loss of learning on task τ becomes:

$$\mathcal{L} = \underbrace{\sum_{u \in \mathbb{V}_\tau} l(\mathrm{MPNN}(\mathcal{G}_u^{sub}; \mathbf{\Theta}), \mathbf{y}_u)}_{\text{loss of the current task } \mathcal{L}_\tau} + \lambda \underbrace{\sum_{\bar{\mathcal{G}}_v^{sub} \in \mathcal{SEM}} l(\mathrm{MPNN}(\bar{\mathcal{G}}_v^{sub}; \mathbf{\Theta}), \mathbf{y}_v)}_{\text{auxiliary loss } \mathcal{L}_{aux}} . \tag{5.6}$$

Unlike in traditional learning which selects λ manually, to overcome the severe class imbalance problem in graph datasets, we choose to balance the loss with the class sizes as shown in Section 6.3.2.3.

TABLE 5.1. Statistics of datasets and task splittings

Dataset	CoraFull [65]	OGB-Arxiv [38]	Reddit [35]	OGB-Products [38]
# nodes	19,793	169,343	232,965	2,449,029
# edges	130,622	1,166,243	114,615,892	61,859,140
# classes	70	40	40	47
# tasks	35	20	20	23

5.3 Experiments

In this section, we aim to answer the research questions: **RQ1**: Whether storing subgraphs (sparsified) guarantees a better performance compared to only storing nodes? **RQ2**: Whether our proposed model can outperform existing state-of-the-art methods in the challenging class-IL scenario?

5.3.1 Datasets

We followed the Continual Graph Learning Benchmark (CGLB) [121] and adopted four large public datasets, including the datasets with up to millions of nodes, hundreds of millions of edges, and tens of tasks, which are very challenging for class-IL scenario. For all datasets, each task contains two classes. For each class, 60% of the data are used for training, 20% are used for validation, and the remaining 20% are used for testing. Detailed dataset statistics are shown in Table 6.1.

5.3.2 Experimental Settings

5.3.2.1 Continual learning setting and model evaluation

In continual learning, a model continuously learns a sequence of tasks with access only to the data of the current task during training. During testing, the model is tested on all learned tasks. The setting is further divided into class-incremental (class-IL) and task-incremental (task-IL) scenario [121]. Class-IL scenario requires a model to classify the given data by picking a class from all previously learnt classes, while the task-IL scenario only requires the

model to distinguish the classes within each task. For example, suppose the model learns on a citation network with a two-task sequence {(*physics*, *chemistry*), (*biology*, *math*)}. In class-IL scenario, after training, the model is required to classify a given document into one of the four classes. In task-IL scenario, the model is only required to classify a document into to (*physics*, *chemistry*) or (*biology*, *math*), while cannot distinguish between *physics* and *biology* or *chemistry* and *math*.

TABLE 5.2. Performance comparisons under class-IL on 4 datasets (↑ higher means better).

Techniques	CoraFull		OGB-Arxiv		Reddit		OGB-Products	
	AA/% ↑	AF/% ↑	AA/% ↑	AF /% ↑	AA/% ↑	AF /% ↑	AA/% ↑	AF /% ↑
Fine-tune	3.5±0.5	-95.2±0.5	4.9±0.0	-89.7±0.4	5.9±1.2	-97.9±3.3	3.4±0.8	-82.5±0.8
EWC [44]	52.6±8.2	-38.5±12.1	8.5±1.0	-69.5±8.0	10.3±11.6	-33.2±26.1	23.8±3.8	-21.7±7.5
MAS [3]	12.3±3.8	-83.7±4.1	4.9±0.0	-86.8±0.6	13.1±2.6	-35.2±3.5	16.7±4.8	-57.0±31.9
GEM [59]	8.4±1.1	-88.4±1.4	4.9±0.0	-89.8±0.3	28.4±3.5	-71.9±4.2	5.5±0.7	-84.3±0.9
TWP [58]	62.6±2.2	-30.6±4.3	6.7±1.5	-50.6±13.2	13.5±2.6	-89.7±2.7	14.1±4.0	-11.4±2.0
LwF [52]	33.4±1.6	-59.6±2.2	9.9±12.1	-43.6±11.9	86.6±1.1	-9.2±1.1	48.2±1.6	-18.6±1.6
ER-GNN [129]	34.5±4.4	-61.6±4.3	30.3±1.5	-54.0±1.3	88.5±2.3	-10.8±2.4	56.7±0.3	-33.3±0.5
Joint	81.2±0.4	-3.3±0.8	51.3±0.5	-6.7±0.5	97.1±0.1	-0.7±0.1	71.5±0.1	-5.8±0.3
SSM-uniform	73.0±0.3	-14.8±0.5	47.1±0.5	-11.7±1.5	94.3±0.1	-1.4±0.1	62.0±1.6	-9.9±1.3
SSM-degree	**75.4±0.1**	-9.7±0.0	**48.3±0.5**	-10.7±0.3	**94.4±0.0**	-1.3±0.0	**63.3±0.1**	-9.6±0.3

For continual learning, the most thorough evaluation is the accuracy matrix $\mathrm{M}^{acc} \in \mathbb{R}^{T\times T}$. Each entry $\mathrm{M}^{acc}_{i,j}$ denotes the model's accuracy on task j after learning task i. Each row $\mathrm{M}^{acc}_{i,:}$ shows the accuracy on all previous tasks after learning task i. Each column $\mathrm{M}^{acc}_{:,j}$ shows how the model's accuracy on task j changes when being trained consecutively on all T tasks. To derive a single numeric value for evaluation, the average accuracy (AA) $\frac{\sum_{i=1}^{T}\mathrm{M}^{acc}_{T,i}}{T}$ and average forgetting (AF) $\frac{\sum_{i=1}^{T-1}\mathrm{M}^{acc}_{T,i}-\mathrm{M}^{acc}_{i,i}}{T-1}$ after learning all T tasks will be used. All experiments are repeated 5 times on an Nvidia Titan Xp GPU. Results are reported with mean and standard deviations.

5.3.2.2 Baselines and model settings

Our baselines are adopted from CGLB [121] including Experience Replay based GNN (ERGNN) [129], Topology-aware Weight Preserving (TWP) [58], Elastic Weight Consolidation (EWC) [44], Learning without Forgetting (LwF) [52], Gradient Episodic Memory (GEM) [59], and

Memory Aware Synapses (MAS) [3]). We also adopt joint training as the upper bound [58, 121, 122]. A jointly trained model is simultaneously trained on all tasks without forgetting problem. Besides, fine-tune (without continual learning technique) is adopted as the lower bound [58, 121, 122]. All models are implemented based on four popular backbone GNNs, *i.e.*, Graph Convolutional Networks (GCNs) [43], Simple Graph Convolution (SGC) [97], Graph Attention Networks (GATs) [87], and Graph Isomorphism Network (GIN) [105]. Overall, the results of different backbones do not exhibit an essential difference in performance. Since the full results on all backbones are too many to be shown and the backbone choice is not our key focus, we show the best results of each method. For a fair comparison, all backbones are set as 2-layer with 256 hidden dimensions.

5.3.2.3 Class imbalance & class-IL classifier

According to Section 6.3.2.1, the performance on different tasks contribute equally to the average accuracy. However, unlike the traditional continual learning with balanced datasets, the class imbalance problem is usually severe in graphs, of which the effect will be entangled with the effect of forgetting. To balance the data by simply choosing an equal number of graph nodes from each class is impractical. For example, in the OGB-Products dataset, the largest class has 668,950 nodes, while the smallest contains only 1 node. Therefore, sampling an equal amount of nodes from each class would result in either deleting many classes without enough nodes or sampling a very small number of nodes from each class so that all classes can provide enough nodes. Moreover, deleting nodes in a graph would also change the original topological structures of the remaining nodes, which is undesired. To this end, we propose to rescale the loss according to the class sizes. Denoting the set of all classes as $\mathcal{C}$, and the number of examples of each class as $\{n_c \mid c \in \mathcal{C}\}$, we can calculate a scale for each class c to balance their contribution in the loss function as $s_c = \frac{n_c}{\sum_{i \in \mathcal{C}} n_i}$. Finally, the balanced loss is:

$$\mathcal{L} = \sum_{v \in \mathbb{V}_\tau} l(f(\mathbf{e}_v; \boldsymbol{\theta}), y_v) \cdot s_{y_v} + \sum_{\mathbf{e}_w \in \mathcal{SEM}} l(f(\mathbf{e}_w; \boldsymbol{\theta}), y_w) \cdot s_{y_w}, \tag{5.7}$$

λ in Equation (6.7) is omitted as it will influence the balance of each class. Empirically, this choice results in significantly better performance for all methods.

The number of the output heads of a model in a standard classification task equals the number of classes and is fixed at the beginning. But in the class-IL, the output heads continually increase with the new classes. To better accommodate the new classes, cosine distance is adopted by some works [98, 91] to modify the standard softmax classifier. In our experiments, all baselines except LwF adopt the standard classifier, since only LwF exhibits better performance through the classifier modified with the cosine distance.

5.3.3 Comparisons for Class-IL Scenario ($\mathrm{RQ1}, \mathrm{RQ2}$)

In this subsection, we compare SEM and the baselines under the class-IL scenario. For SEM, both SEM with uniform sampling (SEM-uniform) and SEM with degree based sampling (SEM-degree) are included. For OGB-Arxiv, OGB-Products, and Reddit datasets, we choose the buffer size to be 400 per class. For CoraFull, we only allow a budget of 60 per class. For the memory based baselines, we allow a budget of up to 800 per class for all datasets to highlight the advantage of SEM. As shown in Table 6.2, under the class-IL setting, SEM not only outperforms the baselines with a large margin on all datasets, but also is comparable to the joint training. In addition, for SEM, degree based sampling also yields better performance than uniform sampling.

5.4 Conclusion

In this work, we developed the Subgraph Episodic Memory (SEM), which stores the topological information in the form of sparsified computation subgraphs to perform continual

graph representation learning. By sampling based graph sparsification, we reduce the memory consumption of a computation subgraph from $\mathcal{O}(d^L)$ to $\mathcal{O}(1)$, and for the first time enable GNNs to fully utilize the explicit topological information for memory replay. Our proposed method outperforms the existing state-of-the-art methods by a large margin on 4 public datasets in the class-IL (more practical and challenging) scenario.

CHAPTER 6

Ricci Curvature based Graph Sparsification for Continual Graph Representation Learning

6.1 Introduction

In this chapter, we propose a Ricci-curvature based graph sparsification technique to obtain the sparsified computation subgraphs for memory-replay based CGL techniques. In real-world graph applications, it is critical to ensure that Graph Neural Networks (GNNs) [43, 32, 35] are capable of continually adapting to new tasks without interfering with their performance over previous tasks. Because of this, continual graph representation learning is attracting increasingly more attention recently. For example, a community detection model is expected to detect the newly emerged communities in a social network while maintaining its capability to recognize existing communities; a document classifier should be able to classify articles belonging to either existing or newly emerged research areas in a citation network, *etc.* However, the rich topological connections among different data samples (graph nodes) pose great challenges to applying some most effective continual learning techniques on graph data.

Memory replay, inspired by research on cognitive science [66, 48], has demonstrated state-of-the-art performance in various classical continual learning tasks. The key idea is to replay a subset of data samples from previous tasks over the model while learning the new tasks [74, 59, 5]. Due to its success, memory replay is also adopted for continual learning on graph data by storing and replaying representative nodes [129, 89]. For graph data, however, only storing a subset of nodes for replay will neglect the important topological information. Since the properties of the target node are not only determined by itself but also by its neighbors, in this

work, we propose to store representative computation subgraphs to explicitly preserve the topological information for memory replay. Directly storing computation subgraphs, however, will trigger the memory explosion problem. Supposing the average node degree is d, then in a L-layer GNN following the message passing paradigm [32], the size of the neighboring nodes in the computation subgraph of a node will be $\mathcal{O}(d^L)$, let alone the associated edges which are typically several orders of magnitude more than the nodes. For instance, in the Reddit-CL dataset, the average node degree is 492, and the maximal degree is 21,657. Obviously, directly storing the entire computation subgraphs is infeasible.

To fully utilize the topological information while maintaining a tractable space complexity at the same time, we propose to sparsify the computation subgraph and preserve the most informative structures (nodes and edges) by utilizing the edge based Ricci curvature [86, 110]. Specifically, the Ricci curvature quantifies how easily information is propagated between two nodes. Given a pair of nodes u and v connected by an edge e_{uv}, the curvature Ric(u,v) is calculated based on the surrounding topological connections like the triangles and 4-cycles based at e_{uv}. Consequently, a higher curvature implies that information could be propagated more easily from u to v and vice versa (as shown in Figure 4.1(b)). In other words, among all edges connected to a node v, the ones with higher curvatures contribute more in generating the representation of v and should be preserved. In contrast, the edges with low curvatures provide little information to v and can be deleted without significant influence on the representation of v. Therefore, to sparsify the computation subgraph of a node v containing its L-hop neighbors, we propose to sample a subset of the edges and their associated nodes based on curvatures. To ensure the connectivity of sparsified subgraphs, we start by sampling the 1-hop neighbors, and then iteratively sample the higher-order neighbors hop by hop. In this way, we can fix the budget for selected edges and nodes, and the size of the sparsified computation subgraph will be independent of the original computation subgraph, *i.e.*, the memory space complexity can be reduced from $\mathcal{O}(d^L)$ to $\mathcal{O}(1)$.

The Ricci curvature has several formulations. In this work, we adopt the Balanced Forman curvature [86] which is theoretically justified to be negatively correlated to the information bottleneck. Considering that the computation complexity of Balanced Forman curvature

is $|\mathbb{E}|d_{\max}^2$, which could be time-consuming to compute on dense graphs, we circumvent computing the exact curvature values by approximating it with the graph diffusion process formulated as a lazy random walk. We theoretically demonstrate that under certain rules, the probability distribution at a node u of a lazy random walk starting at node v is positively correlated to the Ricci curvature between v and u. Therefore, in practice, we sample the edges based on the probability distribution of the random walk and use it as a surrogate for the exact Ricci curvature. Note that we are aware of existing graph sparsification techniques [41, 64, 15, 17, 50, 63, 111, 60, 128, 10], however, most of them are not applicable for continual graph representation learning as detailed in Chapter 2.

In the experiments, we compare Ricci curvature-based sampling with uniform sampling, degree-based sampling [124], and justify its effectiveness. Based on our thorough empirical studies on four different public datasets, SEM with Ricci curvature-based sampling outperforms the existing state-of-the-art methods, especially in the challenging class-IL scenario. Moreover, the performance of SEM is even comparable to joint training which is the performance upper bound for continual learning. To summarize, our contributions are:

(1) We develop the Subgraph Episodic Memory (SEM) to store the explicit topological information in the form of computation subgraphs and perform memory replay based continual graph representation learning.
(2) We resolve the memory explosion problem by sparsifying the subgraphs with Ricci curvature.
(3) We provide a theoretically justified surrogate for Ricci curvature in the sparsification process to ensure its scalability.
(4) Our proposed SEM with Ricci curvature-based sampling outperforms the existing state-of-the-art methods, especially in the challenging class-IL scenario.

6.2 Methods

In this section, we first introduce the preliminaries that include the basic concepts, notations, and learning settings. Next, we explain how memory replay works in traditional continual

learning with independent data examples and why applying memory replay with GNNs on individual graph nodes can result in severe information loss. After that, we explain the memory explosion problem triggered by directly storing the complete topological information. Finally, we introduce our proposed solution for topology sparsification.

6.2.1 Preliminaries

Graph representation learning research focuses on node- or graph-level representations. Graph-level representation learning studies independent graph examples without connections among individual examples. It differs from node-level representation learning, in which the rich topological connections among individual examples have to be carefully considered. In this work, we focus on the node-level continual graph representation learning, which aims to continuously accommodate the new types (classes) of emerging nodes (new tasks) and their associated edges without interfering with the performance over existing nodes (previous tasks). With this setting, a model is trained on a sequence of tasks (subgraphs): $\mathcal{S} = \{\mathcal{G}_1, \mathcal{G}_2, ..., \mathcal{G}_T\}$. Each $\mathcal{G}_\tau$ contains nodes belonging to a unique set of classes in its node set $\mathbb{V}_\tau$. The associated edge set is $\mathbb{E}_\tau$, in which an edge e_{uv} denotes the existence of an edge connecting node u and v. $\mathbb{E}_\tau$ is often represented as the adjacency matrix $\mathbf{A}_\tau \in \mathbb{R}^{|\mathbb{V}_\tau|\times|\mathbb{V}_\tau|}$, where every non-zero entry corresponds to an edge in $\mathbb{E}_\tau$. $\mathbf{A}_\tau$ can be normalized as $\hat{\mathbf{A}}_\tau = \mathbf{D}_\tau^{-\frac{1}{2}}\mathbf{A}_\tau\mathbf{D}_\tau^{-\frac{1}{2}}$, where $\mathbf{D}_\tau \in \mathbb{R}^{|\mathbb{V}_\tau|\times|\mathbb{V}_\tau|}$ is the degree matrix that contains the degree of each node (number of connected edges) in its diagonal entries. Each node $u \in \mathbb{V}_\tau$ has a feature vector $\mathbf{x}_u \in \mathbb{R}^d$, and a label $\mathbf{y}_u \in \{0,1\}^C$, where C is the number of all possible classes. GNNs generate the representation for a node u based on a computation subgraph denoted as $\mathcal{G}_{\tau,u}^{sub}$, which is a subgraph of $\mathcal{G}_\tau$. In the following, $\mathcal{G}_u^{sub}$ without the graph index will be used for simplicity. Finally, we denote the L-hop neighbors of u as $\mathcal{N}^L(u)$ containing the nodes within a distance of L from u, *i.e.*, for a node $u \in \mathbb{V}_i$:

$$\mathcal{N}^L(u) = \{v \in \mathbb{V}_\tau | \mathrm{d}(u,v) = L\}, \tag{6.1}$$

where $\mathrm{d}(\cdot,\cdot)$ denotes the shortest path distance between two nodes. Specially, we have $\mathcal{N}^0(u) = \{u\}$. The vast majority of GNNs follow the message passing neural network

(MPNN) paradigm [32]. In this work, we focus on the MPNNs and refer all GNNs to MPNNs in the following.

6.2.2 Memory Replay on Graphs

In this subsection, we first introduce how memory replay works in traditional continual learning, and then derive the information loss of directly applying memory replay to store individual graph nodes. Finally, we introduce the challenge of storing the topological information of graph data.

Traditional continual learning can be described as training a model $\mathrm{f}(\cdot;\boldsymbol{\theta})$ on a task sequence of length T with the accompanied datasets $\mathbb{D}_\tau = \{(\mathbf{x}_i, \mathbf{y}_i)_{i=1}^{n_\tau}\}$ ($\tau \in \{1, ..., T\}$). The dataset for τ-th task $\mathbb{D}_\tau$ is only available when learning this task, and becomes inaccessible afterward. To alleviate the forgetting problem, memory replay based methods typically maintain a memory buffer $\mathcal{B}$ containing the representative data from the previous tasks, which are replayed to the model when learning new tasks. A straightforward way to utilize $\mathcal{B}$ is through an auxiliary loss:

$$\mathcal{L} = \underbrace{\sum_{\mathbf{x}_i \in \mathbb{D}_\tau} l(f(\mathbf{x}_i;\boldsymbol{\theta}), \mathbf{y}_i)}_{\text{loss of the current task } \mathcal{L}_\tau} + \lambda \underbrace{\sum_{\mathbf{x}_j \in \mathcal{B}} l(f(\mathbf{x}_j;\boldsymbol{\theta}), \mathbf{y}_j)}_{\text{auxiliary loss } \mathcal{L}_{aux}}, \tag{6.2}$$

where the loss function is denoted as $l(\cdot,\cdot)$, and the contribution of auxiliary loss is balanced by λ. Besides directly optimizing an auxiliary loss, there are also other ways to prevent forgetting with the stored data in $\mathcal{B}$. For example, GEM [59] calibrates the gradients of $\mathcal{L}_\tau$ with the gradients of $\mathcal{L}_{aux}$ to prevent increasing the loss on previous tasks ($\mathcal{L}_{aux}$); iCaRL [74] directly uses the representations of the stored data ($\{f(\mathbf{x}_i;\boldsymbol{\theta}), \mathbf{x}_i \in \mathcal{B}\}$) as prototypes to classify new data. For different approaches, we always have to regenerate their representations based on the buffered data. Traditional continual learning deals with independent data samples, and regenerating the representation $f(\mathbf{x}_i;\boldsymbol{\theta})$ only takes $\mathbf{x}_i$ itself as input. Therefore, the memory space complexity for replaying the representation of one independent example (node) is $\mathcal{O}(1)$.

To capture the rich topological information within a graph, the representation of a node not only depends on itself but also depends on its neighboring nodes and the accompanied edges. This will significantly increase the memory space complexity for replaying a graph node. In the following, we take the Message Passing Neural Networks (MPNNs) as an example. The rule for updating the hidden representation of a node u at layer $l+1$ is:

$$\mathbf{m}_u^{l+1} = \sum_{v \in \mathcal{N}^1(u)} \mathrm{M}_l(\mathbf{h}_u^l, \mathbf{h}_v^l, \mathbf{x}_{u,v}^e; \boldsymbol{\theta}_l^{\mathrm{M}}), \tag{6.3}$$

$$\mathbf{h}_u^{l+1} = \mathrm{U}_l(\mathbf{h}_u^l, \mathbf{m}_u^{l+1}; \boldsymbol{\theta}_l^{\mathrm{U}}), \tag{6.4}$$

where $\mathbf{h}_u^l$, $\mathbf{h}_v^l$ denote the hidden representations at layer l, $\mathbf{x}_{uv}^e$ denotes the possible edge features, the function $\mathrm{M}_l(\cdot,\cdot,\cdot;\boldsymbol{\theta}_l^{\mathrm{M}})$ generates message $\mathbf{m}_u^{l+1}$ from neighboring nodes, and the function $\mathrm{U}_l(\cdot,\cdot;\boldsymbol{\theta}_l^{\mathrm{U}})$ aggregates and updates the neighborhood information $\mathbf{m}_u^{l+1}$ into the representation $\mathbf{h}_u^l$ of the target node u. Given a L-layer MPNN, we can simplify the representation of a node u as:

$$\mathbf{h}_u^L = \mathrm{MPNN}(\mathcal{G}_u^{sub}; \boldsymbol{\Theta}), \tag{6.5}$$

where $\mathrm{MPNN}(\cdot;\boldsymbol{\Theta})$ denotes the composition of the message functions and the update functions at all layers. According to the updating rule, in L-layer MPNNs, $\mathcal{G}_u^{sub}$ would contain the L-hop neighbors $\mathcal{N}^L(v)$ and the accompanied edges.

To perform memory replay over graphs, the naive approach which stores individual nodes will lose the important explicit topological information of the computation subgraph $\mathcal{G}_u^{sub}$. A proper way should be to store representative computation subgraphs of existing tasks. However, due to the rich connections among different nodes, the size of different computation subgraphs varies and could be extremely large. Supposing the average node degree of $\mathcal{G}_v^{sub}$ is d, then the expected space complexity of storing its nodes would be $\mathcal{O}(d^L)$ (number of edges not counted yet), which can easily exceed the memory buffer size. For example, the average degree of the Reddit-CL dataset is 492, and the maximal degree is 21,657, which would easily result in intractable memory consumption.

6.2.3 Graph Sparsification via Information Bottlenecks

Intuitively, the nodes and edges in $\mathcal{G}_u^{sub}$ do not contribute equally to the representation of the target node u and we may need to keep the most prominent ones for memory replay. Since GNNs propagate information via edges, different topological connection patterns between two nodes largely determine the amount of information from one node to the other. Specifically, the neighbors with multiple paths to u would contribute more than those with few paths, as shown in Figure 4.1. Mathematically, this *hardness* for information propagation can be described with graph curvature. In this work, we adopt the Balanced Forman curvature [86] (one specific type of Ricci curvature) which is theoretically justified to be negatively correlated with the information bottleneck. Formally, the curvature is defined as:

DEFINITION 6 (Ricci curvature). *Given an edge e_{uv}, its curvature is defined as:*

$$\begin{aligned} Ric(u,v) :=& \frac{1}{d_u} + \frac{1}{d_v} - 2 + 2\frac{|\#_{\triangle}(u,v)|}{\max\{d_u,d_v\}} + \frac{|\#_{\triangle}(u,v)|}{\min\{d_u,d_v\}} \\ &+ \frac{\gamma_{\max}^{-1}}{\max\{d_u,d_v\}}(|\#_{\square}^{u}(u,v)| + |\#_{\square}^{v}(u,v)|), \end{aligned} \tag{6.6}$$

where d_u denotes the degree of a node, $\#_{\triangle}(u,v)$ and $\#_{\square}^{v}(u,v)$ are the higher order paths which will be detailed in the following, and $\gamma_{\max}^{-1}$ normalizes the number of 4-cycles according to how many cycles have shared edges.

(1) $\#_{\triangle}(u,v) := \mathcal{N}^1(u) \cap \mathcal{N}^1(v)$*, i.e., the number of triangles formed with u, v and one of their common 1-hop neighbors.*

(2) $\#_{\square}^{u}(u,v) := \{w \in \mathcal{N}^1(u) | w \neq v, \exists q \in (\mathcal{N}^1(w) \cap \mathcal{N}^1(v)) \backslash \mathcal{N}^1(u)\}$*. $\#_{\square}^{u}(u,v)$ counts the number of 4-cycles based at edge $e_{u,v}$, i.e., the path of length 3 propagating information from u to v. The second predicate excludes the paths of length 2, which are counted in the triangles $\#_{\triangle}(u,v)$.*

(3) $\gamma_{\max} = \max\big\{ \max_{p \in \#_{\square}^{u}(u,v)}\{|\mathcal{N}^1(p) \cap \#_{\square}^{v}(u,v) \backslash \mathcal{N}^1(u)| - 1\}, \max_{q \in \#_{\square}^{v}(u,v)}\{|\mathcal{N}^1(q) \cap \#_{\square}^{u}(u,v) \backslash \mathcal{N}^1(v)| - 1\}\big\}$ *denotes the maximal number of 4-cycles based at edge $e_{u,v}$ traversing a common node.*

Note that although the curvature is defined based on each edge e_{uv}, it does not only count the contribution of e_{uv}. As shown in Figure 4.1(b), $\mathrm{Ric}(u, v)$ describes how easily information can be propagated from u to v based on topological connections of different lengths of paths around e_{uv}. Based on the Ricci curvature, a computation subgraph $\mathcal{G}_u^{sub}$ can be sparsified by selecting the most informative structures for representing the target node u. Specifically, to ensure the connectivity of the sparsified graph, we start by first sampling the 1-hop neighbors $\mathcal{N}^1(u)$ according to the curvature of the edges connecting u and nodes in $\mathcal{N}^1(u)$. Then, we sample the 2-hop neighbors that are connected to the selected 1-hop neighbors according to the curvature of the edges connecting them. By repeating this process, we sample a fixed size subset of nodes from 1-hop to L-hop neighbors. Together with the associated edges, we obtain a sparsified computation subgraph $\bar{\mathcal{G}}_u^{sub}$. However, the above process requires computing the curvature of a large number of edges in $\mathcal{G}_u^{sub}$. For a graph with $|\mathbb{E}|$ edges and maximal node degree as $d_{\max}$, the computation complexity of the curvature calculation for all edges is $|\mathbb{E}|d_{\max}^2$. On dense graphs with large $|\mathbb{E}|$ and $d_{\max}$, this would be intractable.

Fortunately, to sample the nodes, the exact values of the curvature are not essential. Instead, only the relative magnitudes of the curvatures matter, and thus we may derive surrogate metrics whose values are positively correlated to the curvature. Considering that the curvature describes the easiness of information flow on graphs, which is closely related to the graph diffusion process, the following theorem is derived:

THEOREM 4. *Given a graph $\mathcal{G}_v^{sub}$ and the accompanied node set $\mathbb{V}_v$. For a graph (probability) diffusion process starting at v, we denote the probability distribution over a specific node u after i steps as $p^i(v, u)$. Accordingly, at the beginning, $p^0(v, v) = 1$ and $p^0(v, w) = 0$ for $w \in \mathbb{V}_v \backslash \{v\}$. Setting the rule of propagating the probability mass from node v to neighbors as: $\frac{1}{d_v}$ stays at v, and $\frac{1}{d_v}$ is propagated to each neighbor, the probability distribution on each neighbor $u \in \mathcal{N}^1(v)$ after a 3-step diffusion, i.e. $p^3(v, u)$, is positively correlated to* $\mathrm{Ric}(u, v)$.

According to Theorem 4, a higher $p^3(v, u)$ indicates a higher curvature between v and u, *i.e.*, node v has a higher contribution to the representation of u in the message passing. Therefore, the diffusion probability distribution can be utilized as a surrogate for the Ricci curvature

Algorithm 3 Computation subgraph sparsification

Input: $\mathcal{G}_u^{sub}$, node set $\mathbb{V}^{sub}$, edge set $\mathbb{E}^{sub}$, memory budget $\{k_l | l = 1, ..., L\}$.
Output: Sparsified computation subgraph $\bar{\mathcal{G}}_u^{sub}$
Initialize a node set $\mathbb{V} = \{u\}$, an empty edge set $\mathbb{E}$.
for each $l \leftarrow 1$ **to** L **do**
    Initialize a temporary node set $\mathbb{V}_{temp} = \{u\}$
    Initialize a probability set $\mathbb{P} = \{\}$
    for each $v \in \mathbb{V}_{temp}$ **do**
        for each $w \in \mathcal{N}^1(v)$ **do**
            Compute $p^3(w, v)$
            $\mathbb{P} = \mathbb{P} \cup \{p^3(w, v)\}$
        end for each
    end for each
    Sample a set of k_l nodes $\mathbb{V}_{samp}$ according to $\mathbb{P}$
    $\mathbb{V} = \mathbb{V} \cup \mathbb{V}_{samp}$
    for each $v \in \mathbb{V}_{temp}$ **do**
        for each $w \in \mathbb{V}_{samp}$ **do**
            if $e_{w,v} \in \mathbb{E}^{sub}$ **then**
                $\mathbb{E} = \mathbb{E} \cup \{e_{w,v}\}$ ▷ Store the accompanied edges
            end if
        end for each
    end for each
    $\mathbb{V}_{temp} = \mathbb{V}_{samp}$
end for each
Construct $\bar{\mathcal{G}}_u^{sub}$ with $\mathbb{V}$ and $\mathbb{E}$.

when sampling the most informative structures. Specifically, given a computation subgraph $\mathcal{G}_u^{sub}$ containing the neighbors from 1-hop to L-hop, we set a fixed memory budget K as the total number of nodes to sample from each hop. Denoting the budget for the l-th hop as k_l, we have $\sum_{l=1}^{L} k_l = K$. Then the detailed sampling procedure is described in Algorithm 3. The probability computation is shown in the form of for-loop for clarity. In practice, the random walk visiting probability of multiple nodes is implemented in parallel with mature deep learning libraries like Deep Graph Library (DGL).

Since the memory budget K is constant regardless of the graphs or models, the memory space complexity for replaying one node is reduced to $\mathcal{O}(1)$, which is manageable and similar to the traditional continual learning setting. With the sparisified computation subgraphs stored

in the Subgraph Episodic Memory $\mathcal{SEM}$, the loss of learning on task τ becomes:

$$\mathcal{L} = \underbrace{\sum_{u \in \mathbb{V}_\tau} l(\text{MPNN}(\mathcal{G}_u^{sub}; \mathbf{\Theta}), \mathbf{y}_u)}_{\text{loss of the current task } \mathcal{L}_\tau} + \lambda \underbrace{\sum_{\bar{\mathcal{G}}_v^{sub} \in \mathcal{SEM}} l(\text{MPNN}(\bar{\mathcal{G}}_v^{sub}; \mathbf{\Theta}), \mathbf{y}_v)}_{\text{auxiliary loss } \mathcal{L}_{aux}} . \tag{6.7}$$

Unlike in traditional learning which selects λ manually, to overcome the severe class imbalance problem in graph datasets, we choose to balance the loss with the class sizes as shown in Section 6.3.2.3.

6.3 Experiments

In this section, we aim to answer the following three research questions: **RQ1**: Whether storing subgraphs (sparsified) guarantees a better performance compared to only storing nodes? **RQ2**: How does the performance change with different sparsification rates (different amounts of nodes and edges are removed)? **RQ3**: Whether our proposed model can outperform existing state-of-the-art methods in both class-IL and task-IL scenarios? Our codes will be available online upon acceptance.

6.3.1 Datasets

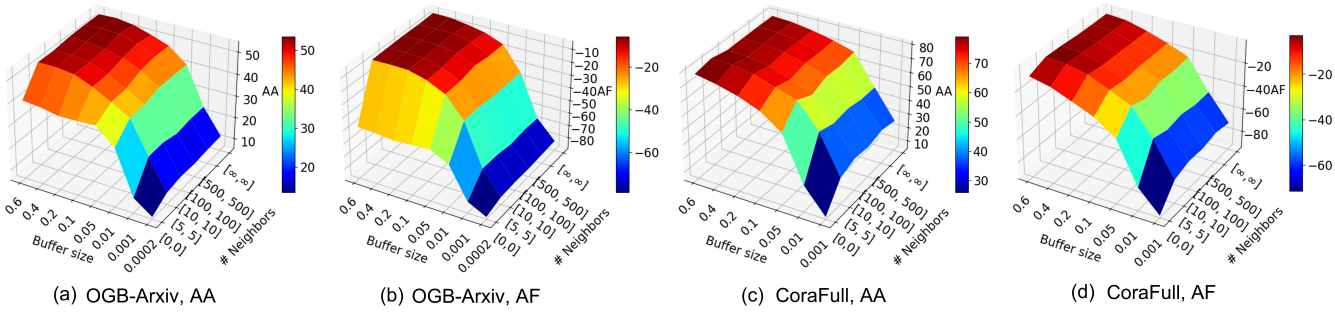
We adopted four large public datasets following the settings of Continual Graph Learning Benchmark (CGLB) [121], including the datasets with up to millions of nodes, hundreds of millions of edges, and tens of tasks, which are very challenging for both class-IL and task-IL scenarios. Among the datasets, CoraFull-CL [65] and Arxiv-CL [39] are citation networks, Reddit-CL is a graph constructed from Reddit posts, and Products-CL [39] is an Amazon product co-purchasing network. For all datasets, each task contains two classes. For each class, 60% of the data are used for training, 20% are used for validation, and the remaining 20% are used for testing. Originally, these datasets were not for continual learning, and were

TABLE 6.1. Statistics of datasets and task splittings

Dataset	CoraFull-CL [65]	Arxiv-CL [38]	Reddit-CL [35]	Products-CL [38]
# nodes	19,793	169,343	232,965	2,449,029
# edges	130,622	1,166,243	114,615,892	61,859,140
# classes	70	40	40	47
# tasks	35	20	20	23

TABLE 6.2. Performance comparisons under class-IL on 4 datasets with SGC backbone (↑ higher means better). The best and the second best performance obtained by continual learning models are highlighted by boldface and underline, respectively.

Techniques	CoraFull		OGB-Arxiv		Reddit		OGB-Products	
	AA/% ↑	AF/% ↑	AA/% ↑	AF /% ↑	AA/% ↑	AF /% ↑	AA/% ↑	AF /% ↑
Fine-tune	3.5±0.5	-95.2±0.5	4.9±0.0	-89.7±0.4	5.9±1.2	-97.9±3.3	7.6±0.7	-88.7±0.8
EWC [44]	52.6±8.2	-38.5±12.1	8.5±1.0	-69.5±8.0	10.3±11.6	-33.2±26.1	23.8±3.8	-21.7±7.5
MAS [3]	6.5±1.5	-92.3±1.5	4.8±0.4	-72.2±4.1	9.2±14.5	-23.1±28.2	16.7±4.8	-57.0±31.9
GEM [59]	8.4±1.1	-88.4±1.4	4.9±0.0	-89.8±0.3	11.5±5.5	-92.4±5.9	4.5±1.3	-94.7±0.4
TWP [57]	62.6±2.2	-30.6±4.3	6.7±1.5	-50.6±13.2	8.0±5.2	-18.8±9.0	14.1±4.0	-11.4±2.0
LwF [52]	33.4±1.6	-59.6±2.2	9.9±12.1	-43.6±11.9	86.6±1.1	-9.2±1.1	48.2±1.6	-18.6±1.6
ER-GNN [129]	34.5±4.4	-61.6±4.3	21.5±5.4	-70.0±5.5	82.7±0.4	-17.3±0.4	48.3±1.2	-45.7±1.3
SSM-uniform [124]	73.0±0.3	-14.8±0.5	47.1±0.5	-11.7±1.5	94.3±0.1	-1.4±0.1	62.0±1.6	-9.9±1.3
SSM-degree [124]	75.4±0.1	-9.7±0.0	48.3±0.5	-10.7±0.3	94.4±0.0	-1.3±0.0	63.3±0.1	-9.6±0.3
Joint	81.2±0.4	-	51.3±0.5	-	97.1±0.1	-	71.5±0.1	-
SEM-curvature	**77.7±0.8**	-10.0±1.2	**49.9±0.6**	-8.4±1.3	**96.3±0.1**	-0.6±0.1	**65.1±1.0**	-9.5±0.8



(a) OGB-Arxiv, AA (b) OGB-Arxiv, AF (c) CoraFull, AA (d) CoraFull, AF

FIGURE 6.1. Influence of buffer size and sparsification levels on the performance of SEM on different datasets. (a) Average accuracy on Arxiv-CL dataset. (b) Average forgetting on Arxiv-CL dataset. (c) Average accuracy on CoraFull-CL dataset. (d) Average forgetting on CoraFull-CL dataset.

proposed by different works. The links to the original sources, as well as detailed dataset statistics are included in Table 6.1.

6.3.2 Experimental Settings

6.3.2.1 Continual learning setting and model evaluation.

In continual learning, a model continuously learns a sequence of tasks. During training, the model has access only to the data of the current task, while during testing, the model is expected to perform well on all previously learned tasks. The setting is further divided into class-incremental (class-IL) and task-incremental (task-IL) scenarios according to whether the task indicators are given.

Suppose the model learns on a citation network with a two-task sequence {(*physics*, *chemistry*), (*biology*, *math*)}. In class-IL scenario, after training, the model is required to classify a given document into one of the four classes. In task-IL scenario, the model is only required to classify a document into to (*physics*, *chemistry*) or (*biology*, *math*), while cannot distinguish between *physics* and *biology* or *chemistry* and *math*. Generally speaking, given a task sequence containing T tasks and each task contains n classes, during testing, task-IL only requires a model to pick a class for a given node from n classes, while class-IL requires a model to pick from $T \times n$ classes. In our experiments, the datasets contain up to 35 tasks and 70 classes. In the case with 35 tasks and each task contains 2 classes, after learning the entire task sequence, class-IL setting requires a model to pick the correct class from 70 classes, while task-IL still only requires the model to distinguish between 2 classes (task identity is given to indicate which two classes are being tested). Considering the difficulty mentioned above together with the forgetting issue, class-IL is more practical and challenging than task-IL. Besides task-IL and class-IL, domain-IL has also been studied and refers to a scenario in which the task is fixed but the domain of the data constantly changes. The difficult of domain-IL is typically between task-IL and class-IL. Existing works typically adopt task-IL or class-IL tasks, which can be naturally constructed from most real-world datasets. While the domain-IL scenario is usually studied for specific applications including knowledge graph [24] and recommender system [1, 106].

For continual learning models, the most thorough evaluation is the accuracy matrix $\mathrm{M}^{acc} \in \mathbb{R}^{T \times T}$. Each entry $\mathrm{M}^{acc}_{i,j}$ denotes the model's accuracy on task j after learning task i. Then

each row $\mathrm{M}^{acc}_{i,:}$ shows the model's accuracy on all previous tasks after learning task i, and each column $\mathrm{M}^{acc}_{:,j}$ shows how the model's accuracy on task j changes when being trained consecutively on all T tasks. To derive a single numeric value for evaluation, the average accuracy (AA) $\frac{\sum_{i=1}^{T}\mathrm{M}^{acc}_{T,i}}{T}$ and average forgetting (AF) $\frac{\sum_{i=1}^{T-1}\mathrm{M}^{acc}_{T,i}-\mathrm{M}^{acc}_{i,i}}{T-1}$ after learning all T tasks will be used. All experiments are repeated 5 times on one Nvidia Titan Xp GPU, and results are reported with average performance and standard deviations.

6.3.2.2 Baselines and model settings

Our adopted baselines include the methods specially designed for continual graph learning: Experience Replay based GNN (ERGNN) [129] and Topology-aware Weight Preserving (TWP) [57], and milestone works designed for traditional continual learning on Euclidean data but also applicable to GNNs: Elastic Weight Consolidation (EWC) [44], Learning without Forgetting (LwF) [52], Gradient Episodic Memory (GEM) [59], and Memory Aware Synapses (MAS) [3]). We also adopt joint training as the upper bound. A jointly trained model is simultaneously trained on all tasks. Therefore, it has no forgetting problem and can serve as an upper bound on the continual learning performance. Besides, fine-tune (without continual learning technique) is adopted as the lower bound. All models are implemented based on four popular backbone GNNs, *i.e.*, Graph Convolutional Networks (GCNs) [43], Simple Graph Convolution (SGC) [97], Graph Attention Networks (GATs) [87], and Graph Isomorphism Network (GIN) [105]. For the backbone models, the input dimension is determined by the dimension of the node attribute vector in the given dataset, and the output dimension is determined by the number of classes in each task. To fairly compare different continual learning techniques, all backbones are set as 2-layer with 256 hidden dimensions. For the multi-head mechanism in GAT, we adopt 8 heads, each of which outputs 32 dimensions. Therefore the hidden dimension of GAT is also 256. For all backbone models, the hidden dimension is validated over 32, 128, 256, and 512 on the validation set, and 256 turns out to be the best option.

6.3.2.3 Class imbalance & class-IL classifier

According to Section 6.3.2.1, the performance on different tasks contribute equally to the average accuracy. However, unlike the traditional continual learning with balanced datasets, the class imbalance problem is usually severe in graphs, of which the effect will be entangled with the effect of forgetting. To balance the data by simply choosing an equal number of graph nodes from each class is impractical. For example, in the Products-CL dataset, the largest class has 668,950 nodes, while the smallest contains only 1 node. Therefore, sampling an equal amount of nodes from each class would result in either deleting many classes without enough nodes or sampling a very small number of nodes from each class so that all classes can provide enough nodes. Moreover, deleting nodes in a graph would also change the original topological structures of the remaining nodes, which is undesired. To this end, we propose to rescale the loss according to the class sizes. Denoting the set of all classes as $\mathcal{C}$, and the number of examples of each class as $\{n_c \mid c \in \mathcal{C}\}$, we can calculate a scale for each class c to balance their contribution in the loss function as $s_c = \frac{n_c}{\sum_{i \in \mathcal{C}} n_i}$. Finally, the balanced loss is:

$$\begin{aligned}\mathcal{L} = &\sum_{v \in \mathbb{V}_\tau} l(f(\mathbf{e}_v; \boldsymbol{\theta}), y_v) \cdot s_{y_v} \\ &+ \sum_{\mathbf{e}_w \in \mathcal{SEM}} l(f(\mathbf{e}_w; \boldsymbol{\theta}), y_w) \cdot s_{y_w},\end{aligned} \tag{6.8}$$

λ in Equation (6.7) is omitted as it will influence the balance of each class. Empirically, this choice results in significantly better performance for all methods.

The number of the output heads of a model in a standard classification task equals the number of classes and is fixed at the beginning. But in the class-IL scenario, the output heads continually increase along with the new classes. To better accommodate the new classes, cosine distance is adopted by a number of works [98, 91, 31] to modify the standard softmax classifier. In our experiments, all baselines except LwF adopt the standard classifier, since only LwF exhibits better performance through the classifier modified with the cosine distance.

6.3.3 Studies on Sparsification Levels and Buffer Sizes ($\mathrm{RQ1}, \mathrm{RQ2}$)

In Figure 6.1, we show the average accuracy and forgetting obtained with different buffer sizes and sparsification levels on OBG-Arxiv-CL and CoraFull-CL datasets. Buffer size (per class) denotes the number of nodes whose sparsified computation subgraphs are stored. For clarity, we covert the buffer size into the ratio of the dataset size. For thorough investigation, we vary the buffer size from 1 node per class (0.02% of the size of Arxiv-CL and 0.1% of CoraFull-CL) to the size of the entire training set (60%). The sparsification level is denoted as the number of neighbors to keep for each hop. Since the backbones are 2-layer, only the 2-hop neighbors are concerned.

To clearly denote the numbers of neighbors we store from neighbors at each hop, we enclose them in a square bracket, [n_1, n_2, ...], where n_i denotes the budget for the i-th hop. When the computation subgraph covers 2-hop neighborhood, there are two entries in the brackets, i.e. [n_1, n_2]. Accordingly, there are two extreme cases. First, [0, 0] denotes the situation when we only store the individual center nodes without considering the neighborhood (topology). Second, [∞, ∞] means that we have unlimited budget for each hop and store the entire computation subgraph. From Figure 6.1, we could see a significant increase in the performance when the buffer starts to store subgraphs ([5, 5]) compared to storing individual nodes ([0, 0]), while the performance gain brought by storing denser computation subgraphs (from [5, 5] to [∞, ∞]) is relatively low. This implies the importance of topological information and indicates the computation subgraphs are highly redundant and our sparsification strategies can effectively preserve the crucial information.

6.3.4 Comparisons for Class-IL Scenario ($\mathrm{RQ1}, \mathrm{RQ3}$)

In this subsection, we compare SEM and the baselines under the class-IL scenario. For Arxiv-CL, Products-CL, and Reddit-CL datasets, we choose the buffer size to be 400 per class. For CoraFull-CL, we only allow a budget of 60 per class. For the memory based baselines, we allow a budget of up to 800 per class for all datasets to highlight the advantage of SEM. As shown in Table 6.2, under the class-IL setting, SEM-curvature not only outperforms the

TABLE 6.3. Performance comparisons under task-IL on 4 datasets with SGC backbone (↑ higher means better). The best and the second best performance obtained by continual learning models are highlighted by boldface and underline, respectively.

Techniques	CoraFull		OGB-Arxiv		Reddit		OGB-Products	
	AA/% ↑	AF/%	AA/% ↑	AF /% ↑	AA/% ↑	AF /% ↑	AA/% ↑	AF /% ↑
Fine-tune	56.0±4.2	-41.0±4.5	56.2±2.6	-36.2±2.6	79.5±24.2	-11.7±4.8	64.4±3.8	-31.1±4.4
EWC [44]	89.8±1.0	-5.1±0.5	71.5±0.6	-0.9±0.6	83.9±15.1	-2.0±1.5	87.0±1.4	-1.7±1.2
MAS [3]	92.2±0.9	-3.7±1.3	72.7±2.6	-18.5±2.5	61.1±7.1	-0.5±1.0	80.6±4.3	-13.7±3.7
GEM [59]	91.5±0.5	-1.9±0.9	81.1±1.7	-4.0±1.8	98.9±0.1	-0.5±0.1	87.7±1.8	-7.0±2.0
TWP [57]	94.3±0.9	-1.6±0.4	<u>89.4±0.4</u>	0.0±0.3	78.0±18.5	-0.2±0.4	81.8±3.3	-0.3±0.8
LwF [52]	93.8±0.1	-0.4±0.1	71.1±3.2	-1.5±0.8	98.6±0.1	-0.0±0.0	86.3±0.2	-0.5±0.1
ER-GNN [129]	86.3±1.0	-9.2±0.9	86.4±0.3	0.5±0.6	97.4±0.2	4.7±0.1	86.4±0.0	11.7±0.0
SSM-uniform [124]	95.3±0.5	0.2±0.5	88.5±0.6	-1.3±0.5	99.2±0.0	-0.2±0.0	93.1±0.8	-1.8±0.3
SSM-degree [124]	<u>95.8±0.3</u>	0.6±0.2	88.4±0.3	-1.1±0.1	<u>99.3±0.0</u>	-0.2±0.0	<u>93.2±0.7</u>	-1.9±0.0
Joint	95.5±0.2	-	90.3±0.4	-	99.5±0.0	-	95.3±0.8	-
SEM-curvature	**95.9±0.5**	0.7±0.4	**89.9±0.3**	-0.1±0.5	**99.3±0.0**	-0.2±0.0	**93.2±0.7**	-1.8±0.4

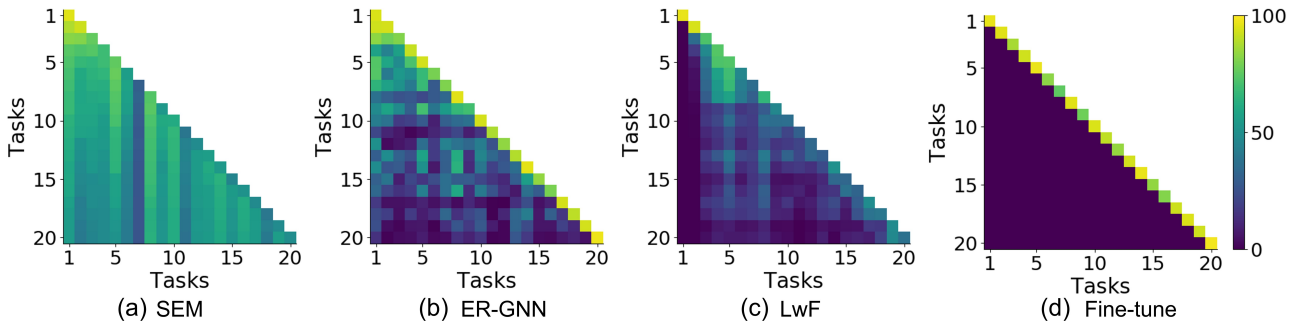


FIGURE 6.2. From left to right: accuracy matrix of SEM, ER-GNN, LwF, and Fine-tune on Arxiv-CL dataset. In an accuracy matrix, an entry at the i-th row and j-th column is the model's accuracy on the j-th task after learning the i-th task.

baselines with a large margin on all datasets, but also outperforms the two memory replay baselines based on uniform sampling and degree based sampling. The performance is even comparable to the joint training. To understand the learning dynamics, we visualize the accuracy matrices of the most representative methods including SEM, the two best baseline methods ER-GNN and LwF (based on knowledge distillation), and fine-tune. As shown in Figure 6.2, by comparing the columns of different matrices, we can find different forgetting patterns. First, SEM maintains a very stable performance of each task throughout the entire training process. The performance exhibits no abrupt decrease compared to the baselines.

Second, the performance of ER-GNN on each task is not monotonically decreasing. Instead, the performance on a task may first decrease and experience a resurgence later. This may be the benefit of the memory buffer, which keeps driving the model to a favorable point for previous tasks. In contrast, the performance of LwF decreases monotonically. Fine-tuning a model in the class-IL scenario is ineffective as the knowledge of previous tasks is completely overwritten by new tasks.

6.3.5 Comparisons for Task-IL Scenario ($\mathrm{RQ1}, \mathrm{RQ3}$)

Finally, we also compare SEM with different baseline approaches in the task-IL scenario. As explained in Section 6.3.2.1, during testing on each task, class-IL requires the model to pick a class from all learnt classes, while task-IL only require the model to distinguish between the classes within the given task. Therefore, class-IL is much more challenging especially when the number of classes are large. In our experiments, the datasets contain at least 40 classes (OGB-Arxiv and Reddit) and at most 70 classes (CoraFull), which significantly increase the learning difficulty of class-IL. Accordingly, as shown in Table 6.3, task-IL is experimentally much less challenging than class-IL, and many more methods perform very well in this scenario.

However, our method still outperforms all baseline techniques on 4 different datasets. Moreover, on all datasets, our method obtains comparable performance with joint training (Joint). Considering that joint training is simultaneously trained on all tasks with full topological information (without the forgetting issue), and is the upper bound on the continual learning models, the effectiveness of our method is very significant.

6.3.6 Performance of Different Backbones

As mentioned before, our SEM-curvature can be implemented with different GNN backbones, and we have adopted 4 popular backbones, Graph Convolutional Networks (GCNs) [43], Simple Graph Convolution (SGC) [97], Graph Attention Networks (GATs) [87], and Graph

Isomorphism Network (GIN) [105]. In this section, we show the results of SEM-curvature with each of the other backbones.

TABLE 6.4. Performance comparisons under class-IL on 4 datasets with GCN backbone (↑ higher means better).

C.L.T.	CoraFull-CL		Arxiv-CL		Reddit-CL		Products-CL	
	AP/% ↑	AF/% ↑	AP/% ↑	AF /% ↑	AP/% ↑	AF /% ↑	AP/% ↑	AF /% ↑
Bare model	2.9±0.0	-94.7±0.1	4.9±0.0	-87.0±1.5	5.1±0.3	-94.5±2.5	3.4±0.8	-82.5±0.8
EWC [44]	15.2±0.7	-81.1±1.0	4.9±0.0	-88.9±0.3	10.6±1.5	-92.9±1.6	3.3±1.2	-89.6±2.0
MAS [3]	12.3±3.8	-83.7±4.1	4.9±0.0	-86.8±0.6	13.1±2.6	-35.2±3.5	15.0±2.1	-66.3±1.5
GEM [59]	7.9±2.7	-84.8±2.7	4.8±0.5	-87.8±0.2	28.4±3.5	-71.9±4.2	5.5±0.7	-84.3±0.9
TWP [57]	20.9±3.8	-73.3±4.1	4.9±0.0	-89.0±0.4	13.5±2.6	-89.7±2.7	3.0±0.7	-89.7±1.0
LwF [52]	2.0±0.2	-95.0±0.2	4.9±0.0	-87.9±1.0	4.5±0.5	-82.1±1.0	3.1±0.8	-85.9±1.4
ER-GNN [129]	3.0±0.1	-93.8±0.5	30.3±1.5	-54.0±1.3	88.5±2.3	-10.8±2.4	24.5±1.9	-67.4±1.9
SSM-uniform [124]	72.3±0.8	-15.5±1.5	45.1±1.1	-12.2±1.4	93.8±0.4	-2.0±0.3	61.8±1.2	-10.7±1.1
SSM-degree [124]	74.4±0.2	-9.9±0.1	46.0±0.4	-11.3±0.8	94.0±0.2	-1.9±0.2	62.9±0.4	-10.3±0.5
Joint	80.6±0.3	-	46.4±1.4	-	99.3±0.2	-	71.5±0.7	-
SEM-curvature	**79.6±0.3**	-2.7±0.1	**51.0±0.3**	-6.7±0.6	**95.1±0.6**	-1.5±0.7	**64.0±1.6**	-11.2±1.8

TABLE 6.5. Performance comparisons under class-IL on 4 datasets with GAT backbone (↑ higher means better). GAT

C.L.T.	CoraFull-CL		Arxiv-CL		Reddit-CL		Products-CL	
	AP/% ↑	AF/% ↑	AP/% ↑	AF /% ↑	AP/% ↑	AF /% ↑	AP/% ↑	AF /% ↑
Bare model	3.1±0.2	-95.4±0.1	4.9±0.0	-89.2±0.9	4.9±0.4	-93.7±1.8	10.9±1.5	-85.4±1.4
EWC [44]	19.8±1.3	-78.3±2.1	4.9±0.0	-88.6±0.8	5.6±0.5	-40.2±3.1	25.1±2.2	-53.3±4.7
MAS [3]	4.5±0.7	-94.0±2.3	5.3±0.4	-77.3±1.9	4.9±0.1	-16.8±1.4	16.5±0.7	-77.5±3.4
GEM [59]	10.9±1.2	-85.7±2.8	4.9±0.0	-90.4±2.4	8.0±1.1	-95.1±3.2	4.8±0.5	-86.2±1.8
TWP [57]	52.2±1.7	-44.7±2.4	7.2±0.8	-83.8±2.3	3.1±0.3	-31.2±1.5	14.1±2.1	-11.4±1.3
LwF [52]	2.9±0.0	-95.6±0.2	4.9±0.0	-89.4±0.0	4.8±0.0	-93.5±1.6	3.2±0.2	-88.5±1.3
ER-GNN [129]	4.0±0.2	-95.1±0.3	4.9±0.0	-88.6±3.3	14.4±0.9	-88.7±2.2	11.4±1.2	-84.0±0.8
SSM-uniform [124]	69.8±1.2	-18.3±1.3	43.5±1.2	-13.5±1.6	91.5±1.4	-3.8±0.6	58.8±1.0	-12.9±1.5
SSM-degree [124]	70.7±1.1	-17.6±1.2	44.8±0.9	-14.4±2.3	93.2±1.4	-5.1±1.3	60.3±0.3	-11.7±0.5
Joint	82.7±0.2	-	44.3±0.2	-	99.0±0.3	-	70.4±0.9	-
SEM-curvature	**81.2±0.1**	-1.1±0.5	**43.3±0.0**	17.1±0.1	**98.4±0.0**	0.3±0.1	**62.2±0.6**	-9.7±1.4

As demonstrated in Table 6.4 to 6.9, SEM-curvature consistently outperform the baselins with different backbones under different incremental scenarios, especially the class-IL scenario.

6.3.7 Performance of Backbones with Different Depths

In this section, we explore how the depth of the backbone model influence the performance. We show the performance of three methods including the Fine-tune as the lower bound, our

TABLE 6.6. Performance comparisons under class-IL on 4 datasets with GIN backbone (↑ higher means better).

Techniques	CoraFull		OGB-Arxiv		Reddit		OGB-Products	
	AA/% ↑	AF/%	AA/% ↑	AF /% ↑	AA/% ↑	AF /% ↑	AA/% ↑	AF /% ↑
Fine-tune	2.9±0.0	-95.3±0.0	4.9±0.0	-87.8±0.0	5.0±0.2	-94.8±1.9	3.1±0.5	-83.1±2.0
EWC [44]	2.9±0.0	-95.1±0.0	4.9±0.0	-88.8±0.0	9.8±2.2	-92.5±1.7	3.2±1.1	-90.5±0.9
MAS [3]	3.0±0.0	-93.8±0.0	4.8±0.0	-87.5±0.0	12.0±1.7	-40.9±2.4	15.7±2.2	-65.4±1.5
GEM [59]	4.1±0.0	-92.9±0.0	4.9±0.0	-88.8±0.0	13.2±1.4	-86.2±4.6	4.7±0.4	-85.1±1.3
TWP [57]	2.9±0.0	-95.2±0.0	4.9±0.0	-88.7±0.0	11.6±1.1	-88.3±2.2	4.3±0.9	-79.9±1.2
LwF [52]	2.9±0.0	-95.1±0.0	4.9±0.0	-87.3±0.0	4.4±0.2	-82.5±1.6	3.1±0.4	-88.9±2.5
ER-GNN [129]	3.1±0.5	-93.9±0.4	21.5±3.3	-70.0±3.4	80.7±1.2	-19.8±1.3	21.3±0.9	-70.2±1.4
SSM-uniform [124]	73.0±0.3	-14.8±0.4	48.3±0.5	-10.6±0.9	91.9±0.7	-3.1±1.2	60.2±0.8	-11.0±1.2
SSM-degree [124]	77.2±0.2	-10.0±0.3	48.9±0.3	-9.0±0.4	92.1±0.8	-2.2±1.0	61.1±1.1	-11.7±0.8
Joint	81.2±0.3	-3.3±0.4	51.5±0.5	-14.8±0.6	97.3±0.5	-	71.1±0.2	-
SEM-curvature	**79.4±0.0**	-4.5±0.0	**49.9±0.4**	-8.4±0.8	**94.0±0.6**	-1.8±0.9	**63.2±0.8**	-10.1±1.9

TABLE 6.7. Performance comparisons under task-IL on 4 datasets with GCN backbone (↑ higher means better).

C.L.T.	CoraFull-CL		Arxiv-CL		Reddit-CL		Products-CL	
	AP/% ↑	AF/% ↑	AP/% ↑	AF /% ↑	AP/% ↑	AF /% ↑	AP/% ↑	AF /% ↑
Bare model	58.0±1.7	-38.4±1.8	61.7±3.8	-28.2±3.3	73.6±3.5	-26.9±3.5	67.6±1.6	-25.4±1.6
EWC [44]	78.9±2.4	-15.5±2.3	78.8±2.7	-5.0±3.1	91.5±4.2	-8.1±4.6	90.1±0.3	-0.7±0.3
MAS [3]	93.0±0.5	-0.6±0.2	88.4±0.2	-0.0±0.0	98.6±0.5	-0.1±0.1	91.2±0.6	-0.5±0.2
GEM [59]	91.6±0.6	-1.8±0.6	87.3±0.6	2.8±0.3	91.6±5.6	-8.1±5.8	87.8±0.5	-2.9±0.5
TWP [57]	92.2±0.5	-0.9±0.3	86.0±0.8	-2.8±0.8	87.4±3.8	-12.6±4.0	90.3±0.1	-0.5±0.1
LwF [52]	56.1±2.0	-37.5±1.8	84.2±0.5	-3.7±0.6	80.9±4.3	-19.1±4.6	66.5±2.2	-26.3±2.3
ER-GNN [129]	90.6±0.1	-3.7±0.1	86.7±0.1	11.4±0.9	98.9±0.0	-0.1±0.1	89.0±0.4	-2.5±0.3
SSM-uniform [124]	94.5±0.8	-0.1±0.9	88.8±1.2	-2.1±0.7	98.8±0.3	-0.9±0.6	90.9±2.8	-2.2±1.0
SSM-degree [124]	94.3±1.1	0.9±0.5	88.1±1.3	-1.0±0.4	99.0±0.2	-0.4±0.2	91.1±0.9	-1.8±0.8
Joint	95.2±0.2	-	90.3±0.2	-	99.4±0.1	-	91.8±0.2	-
SEM-curvature	**95.0±0.5**	-0.2±0.8	**88.8±0.1**	-1.0±0.2	**99.2±0.1**	-0.1±0.3	**91.6±0.5**	-1.5±0.8

proposed SEM-curvature, and the Joint as the upper bound. The backbone is chosen as the GCN model, and the depth varies from 1-layer to 5-layer. We choose the class-IL scenario on Arxiv-CL as the tasks.

As shown in Table 6.10, the performance is good when the model is not very deep (1- or 2-layer). When the depth increases to 3-layer, all methods, especially the Joint, exhibit performance decrease. In this stage, SEM-curvature actually show significant superiority over Joint. The potential reason is that Joint training uses the full historical data with full topology, which is more prone to over-fitting and over-smoothing. In contrast, SEM-curvature greatly

TABLE 6.8. Performance comparisons under task-IL on 4 datasets with GAT backbone (↑ higher means better).

Techniques	CoraFull		OGB-Arxiv		Reddit		OGB-Products	
	AA/% ↑	AF/%	AA/% ↑	AF /% ↑	AA/% ↑	AF /% ↑	AA/% ↑	AF /% ↑
Fine-tune	53.5±0.1	-43.4±0.5	57.9±1.7	-34.5±2.1	79.5±13.0	-11.7±2.9	64.4±2.3	-31.1±2.6
EWC [44]	87.2±0.0	-7.7±0.3	75.1±0.5	0.1±0.2	83.9±11.9	-2.0±1.0	87.0±0.9	-1.7±0.9
MAS [3]	92.2±0.6	-3.7±0.8	72.6±1.9	-19.0±2.2	61.1±4.3	-0.5±0.7	80.6±2.4	-13.7±2.4
GEM [59]	92.0±0.1	-2.7±0.3	79.2±0.4	-5.7±0.3	88.6±4.7	-9.8±2.9	84.3±1.1	-5.4±0.4
TWP [57]	94.2±0.5	-1.4±0.0	86.6±0.2	-1.5±0.3	78.0±13.4	-0.2±0.3	81.8±2.2	-0.3±0.5
LwF [52]	55.1±1.6	-41.8±1.4	54.5±0.7	-36.9±0.8	80.7±4.4	-20.3±6.2	64.3±3.1	-28.4±3.4
ER-GNN [129]	85.4±0.7	-11.1±0.8	85.8±0.1	-1.2±0.1	97.5±1.5	2.6±3.7	86.4±0.0	11.7±0.0
SSM-uniform [124]	91.4±2.3	-1.2±0.4	85.7±2.1	-4.8±1.4	95.8±2.5	-1.7±1.3	89.3±0.5	-3.3±2.1
SSM-degree [124]	92.1±1.7	-1.3±1.0	86.2±1.5	-3.6±1.8	96.0±1.9	-1.8±0.5	90.0±1.3	-3.2±0.6
Joint	95.3±0.1	-	88.8±0.4	-	98.1±0.5	-	90.3±0.2	-
SEM-curvature	**94.5±0.7**	0.5±0.9	**88.0±0.6**	0.2±1.1	**98.0±1.0**	-0.3±0.4	**90.1±0.4**	-0.3±0.7

TABLE 6.9. Performance comparisons under task-IL on 4 datasets with GIN backbone (↑ higher means better).

Techniques	CoraFull		OGB-Arxiv		Reddit		OGB-Products	
	AA/% ↑	AF/%	AA/% ↑	AF /% ↑	AA/% ↑	AF /% ↑	AA/% ↑	AF /% ↑
Fine-tune	57.9±2.0	-38.5±2.3	65.1±1.6	-23.9±1.5	72.2±1.7	-28.9±2.8	65.2±1.8	-27.2±2.1
EWC [44]	69.4±2.1	-25.5±1.6	63.2±2.9	-21.5±3.2	90.8±3.3	-10.6±3.7	88.8±1.6	-0.9±0.2
MAS [3]	93.9±0.1	-0.3±0.1	87.9±0.4	-0.1±0.1	96.7±1.2	-0.8±1.7	87.8±3.2	-1.2±2.1
GEM [59]	90.5±0.9	-2.1±1.3	80.6±0.8	-3.6±1.1	87.6±2.7	-7.4±2.9	88.4±0.0	-2.5±0.4
TWP [57]	92.8±0.1	-1.6±0.2	88.3±0.6	-0.6±0.8	81.2±2.0	-10.0±3.3	83.9±1.3	-0.9±1.4
LwF [52]	59.2±0.9	-37.4±0.9	84.0±1.7	-3.7±1.3	78.4±0.2	-18.8±3.5	63.2±0.5	-29.6±1.8
ER-GNN [129]	92.4±0.1	1.4±0.1	**89.3±0.1**	5.0±0.4	94.1±1.7	-1.5±2.6	86.3±1.4	-4.3±2.5
SSM-uniform [124]	95.3±0.2	0.4±0.6	88.5±0.3	-1.3±0.3	92.1±0.5	-2.5±2.1	88.7±2.4	-3.7±2.1
SSM-degree [124]	95.8±0.3	0.6±0.2	88.4±0.3	-1.1±0.1	92.9±0.6	-2.1±1.4	89.1±1.3	-3.5±1.1
Joint	95.5±0.1	-0.2±0.1	88.9±0.1	1.2±0.4	98.0±0.3	-	89.8±0.3	-
SEM-curvature	**95.9±0.3**	0.7±0.3	88.4±0.2	-1.2±0.4	**97.6±1.3**	-1.6±1.2	**89.5±1.7**	-9.8±1.6

sparsifies the stored graph topology. On the one hand, graph sparsification could effectively alleviate the over-smoothing problem, which is justified by DropEdge [75]. On the other hand, only selecting a subset of historical data could alleviate over-fitting. When the depth further increases, the performance of all methods keeps going down. Joint even exhibits model collapsing when the depth reaches 4-layer. Above all, we could see that SEM-curvature is actually promising with deep GNNs and large receptive fields.

TABLE 6.10. Performance of SEM-curvature with GCN backbone of different depths.

# layers	1		2		3		4	
	AA/% ↑	AF/% ↑	AA/% ↑	AF /% ↑	AA/% ↑	AF /% ↑	AA/% ↑	AF /% ↑
Fine-tune	4.9±0.0	-90.0±0.0	4.9±0.0	-87.6±0.2	4.8±0.0	-75.0±6.4	3.6±0.8	-67.7±5.6
SEM	49.4±0.0	0.8±0.2	51.0±0.3	-6.7±0.6	48.6±0.7	-6.8±0.3	39.4±1.4	-12.9±1.1
Joint	48.1±0.2	-16.0±0.3	46.3±0.5	-	30.1±0.6	-	9.0±1.5	-

# layers	5	
	AA/% ↑	AF/% ↑
Fine-tune	2.5±0.0	-29.8±10.9
SEM	2.5±0.0	-29.4±0.6
Joint	6.2±0.5	-

TABLE 6.11. Meta-paths from v to u via a random walk of 3 moves

Meta-path	1	2	3	4	5	6	7	8	9	10
Start	v	v	v	v	v	v	v	v	v	v
Intermediate node 1	v	v	u	u	v	u	p	p	p	p
Intermediate node 2	v	u	v	u	p	p	v	u	p	q
End	u	u	u	u	u	u	u	u	u	u

6.4 Proof Details

In this section, we theoretically analyze the correlation between the graph diffusion process and the curvature under the lazy random walk framework.

PROOF. Since the diffusion process is modeled as a lazy random walk model in which the probability of staying at a node v is defined as $\frac{1}{d_v}$.To calculate the probability of starting at a v and arriving at another node u after 3 moves, we first divide the walks into 10 meta-paths in Table 6.11. Totally, we have 10 different paths. v and u are always the starting and ending nodes, and the choice of the two intermediate nodes could be different. In Table 6.11, p and q denote two nodes other than v and u. Since p and q do not refer to specific nodes, some seemingly different meta-paths actually refer to a same meta-path and are not listed, *e.g.* path $vpqu$ and $vqpu$, path $vppu$ and $vqqu$, *etc.*

Then we calculate the probability of walking from v to u via each meta-path. In our random walk process, the self-loop is added to each node and at each step the probability to not move is 0. In other words, a path vv does not imply that the walker stays at the original node, instead it chooses the self-loop of v. Since the probability of choosing each neighbor is equal, this setting ensures the probability of taking path vv to be $\frac{1}{d_v}$. Note that this does not include v into $\mathcal{N}^1(v)$, since $d_G(v, v) = 0$, and $v \in \mathcal{N}^0(v)$. And we have $d_v - 1 = |\mathcal{N}^1(v)|$. Besides, we assume the graph to be a simple, *i.e.* two nodes can be connected via no more than one edge. The probabilities of each meta-path are:

$$p(vvvu) = \frac{1}{d_v} \cdot \frac{1}{d_v} \cdot \frac{1}{d_v} = \frac{1}{d_v^3} \tag{6.9}$$

$$p(vvuu) = \frac{1}{d_v} \cdot \frac{1}{d_v} \cdot \frac{1}{d_u} = \frac{1}{d_v^2 \cdot d_u} \tag{6.10}$$

$$p(vuvu) = \frac{1}{d_v} \cdot \frac{1}{d_u} \cdot \frac{1}{d_v} = \frac{1}{d_v^2 \cdot d_u} \tag{6.11}$$

$$p(vuuu) = \frac{1}{d_v} \cdot \frac{1}{d_u} \cdot \frac{1}{d_u} = \frac{1}{d_v \cdot d_u^2} \tag{6.12}$$

$$\begin{aligned} p(vvpu) &= \sum_{p \in \mathcal{N}^1(v) \cap \mathcal{N}^1(u)} \frac{1}{d_v} \cdot \frac{1}{d_v} \cdot \frac{1}{d_p} \\ &= \sum_{p \in \mathcal{N}^1(v) \cap \mathcal{N}^1(u)} \frac{1}{d_v^2 \cdot d_p} \end{aligned} \tag{6.13}$$

$$\begin{aligned} p(vupu) &= \sum_{p \in \mathcal{N}^1(u)} \frac{1}{d_v} \cdot \frac{1}{d_u} \cdot \frac{1}{d_p} \\ &= \sum_{p \in \mathcal{N}^1(u)} \frac{1}{d_v \cdot d_u \cdot d_p} \end{aligned} \tag{6.14}$$

$$p(vpvu) = \sum_{p \in \mathcal{N}^1(v)} \frac{1}{d_v} \cdot \frac{1}{d_p} \cdot \frac{1}{d_v} = \sum_{p \in \mathcal{N}^1(v)} \frac{1}{d_v^2 \cdot d_p} \tag{6.15}$$

$$\begin{aligned} p(vpuu) &= \sum_{p \in \mathcal{N}^1(v) \cap \mathcal{N}^1(u)} \frac{1}{d_v} \cdot \frac{1}{d_p} \cdot \frac{1}{d_u} \\ &= \sum_{p \in \mathcal{N}^1(v) \cap \mathcal{N}^1(u)} \frac{1}{d_v \cdot d_u \cdot d_p} \end{aligned} \tag{6.16}$$

$$\begin{aligned} p(vppu) &= \sum_{p \in \mathcal{N}^1(v) \cap \mathcal{N}^1(u)} \frac{1}{d_v} \cdot \frac{1}{d_p} \cdot \frac{1}{d_p} \\ &= \sum_{p \in \mathcal{N}^1(v) \cap \mathcal{N}^1(u)} \frac{1}{d_v \cdot d_p^2} \end{aligned} \tag{6.17}$$

$$p(vpqu) = \sum_{p \in \mathcal{N}^1(v), q \in (\mathcal{N}^1(p) \cap \mathcal{N}^1(u) \setminus \{v\})} \frac{1}{d_v} \cdot \frac{1}{d_p} \cdot \frac{1}{d_q} \tag{6.18}$$

The Equation 6.9 to 6.12 correspond to the paths only concerned with Summing up the Equations 6.9 to 6.12, Equations 6.14 and 6.15 as p_1, we get:

$$p_1 = \frac{1}{d_v} \cdot (\frac{1}{d_v^2} + \frac{2}{d_v d_u} + \frac{1}{d_u^2} + \frac{|\mathcal{N}^1(u)|}{d_u d_p} + \frac{|\mathcal{N}^1(v)|}{d_v d_p}) \tag{6.19}$$

$$\begin{aligned} &= \frac{1}{d_v} \cdot (\frac{1}{d_v^2} + \frac{1}{d_v d_u} + \frac{|\mathcal{N}^1(v)|}{d_v d_p}) \\ &+ \frac{1}{d_u} \cdot (\frac{1}{d_v^2} + \frac{1}{d_v d_u} + \frac{|\mathcal{N}^1(v)|}{d_v d_p}) \end{aligned} \tag{6.20}$$

Since both $(\frac{1}{d_v^2} + \frac{1}{d_v d_u} + \frac{|\mathcal{N}^1(v)|}{d_v d_p})$ and $(\frac{1}{d_v^2} + \frac{1}{d_v d_u} + \frac{|\mathcal{N}^1(v)|}{d_v d_p})$ are positive, the two terms of p_1 are positively correlated to the first two terms $\frac{1}{d_v}$ and $\frac{1}{d_u}$ of Equation 6.6, respectively. Whenever $\frac{1}{d_v}$ or $\frac{1}{d_u}$ increases, their corresponding part in p_1 would increase. Similarly, we will find the correspondence between the rest terms of Equation 6.6 and the random walk probabilities. Equations 6.13, 6.16, and 6.17 depends on the common neighbors of u and v, which corresponds to the triangle counter $\#_\triangle$ in Equation 6.6 By summing up these equations,

we get:

$$p_2 = \sum_{p \in \mathcal{N}^1(v) \cap \mathcal{N}^1(u)} \frac{1}{d_v^2 \cdot d_p} + \sum_{p \in \mathcal{N}^1(v) \cap \mathcal{N}^1(u)} \frac{1}{d_v \cdot d_u \cdot d_p} \tag{6.21}$$

$$+ \sum_{p \in \mathcal{N}^1(v) \cap \mathcal{N}^1(u)} \frac{1}{d_v \cdot d_p^2} \tag{6.22}$$

$$= \sum_{p \in \mathcal{N}^1(v) \cap \mathcal{N}^1(u)} \frac{1}{d_p} \cdot (\frac{1}{d_v^2} + \frac{1}{d_v \cdot d_u} + \frac{1}{d_v \cdot d_p}) \tag{6.23}$$

By definition, $|\mathcal{N}^1(v) \cap \mathcal{N}^1(u)| = |\#_{\triangle}(u, v)|$. Since all summation terms in Equation 6.21 are positive, we can rewrite it in a more concise format by assuming all nodes in $\mathcal{N}^1(v) \cap \mathcal{N}^1(u)$ has a degree $d_p = \bar{d_p}$:

$$p_2 = |\#_{\triangle}(v, u)| \cdot \frac{1}{\bar{d_p}} \cdot (\frac{1}{d_v^2} + \frac{1}{d_v \cdot d_u} + \frac{1}{d_v \cdot \bar{d_p}}) \tag{6.24}$$

And the corresponding terms in Equation 6.6 is:

$$|\#_{\triangle}(u, v)| \cdot (\frac{2}{\max\{d_u, d_v\}} + \frac{1}{\min\{d_u, d_v\}}). \tag{6.25}$$

Since both expressions have positive coefficients on the term $|\#_{\triangle}(u, v)|$, they are also positively correlated. Finally, by definition, $|\#_{\square}^{u}(u, v)|$ or $|\#_{\square}^{v}(u, v)|$ refer to the number of 1-hop neighbors of u or v participating in a 4-cycle based at edge $e_{u,v}$. Therefore, the number of 4-cycles based at $e_{u,v}$ is $|\#_{\square}(u, v)| = \max(|\#_{\square}^{u}(u, v)|, |\#_{\square}^{v}(u, v)|)$. By observing Equation 6.18, by traversing p, q via $p \in \mathcal{N}^1(v), q \in (\mathcal{N}^1(p) \cap \mathcal{N}^1(u) \backslash \{v\})$, we are also counting the number of 4-cycles, **i.e.** $|\#_{\square}(u, v)|$. □

6.5 Conclusion

In this work, we developed Ricci curvature based graph sparsification technique to perform continual graph representation learning. We employed Subgraph Episodic Memory (SEM) to store the topological information in the form of computation subgraphs. By leveraging Ricci curvature to assess the informativeness, we can reduce the memory consumption of a computation subgraph from $\mathcal{O}(d^L)$ to $\mathcal{O}(1)$, and enable GNNs to utilize the most informative topological information for memory replay. Our proposed method outperforms the existing

state-of-the-art methods on 4 public datasets in both class-IL (more practical and challenging) and task-IL scenarios.

CHAPTER 7

Conclusions

This chapter first conclude the works already done in this thesis, and then discuss potential perspectives for future research.

7.1 Summary of Contributions

In this thesis, our first contribution is to systematically analyze the setup of continual graph learning and proposed the CGLB to thoroughly and fairly assess state-of-the-art methods from different perspectives, as well as the toolkit for developing new CGL techniques. CGLB contains N-CGL and G-CGL under both task-IL and class-IL scenarios. Besides, we also tested several representative existing techniques on CGLB with detailed analysis with the model performance. By providing the CGLB as well as the toolkit for training, evaluation, and visualization, we greatly lower the barrier to explore CGL problem. In the future, we will also keep adopting new datasets and new benchmark results into CGLB. We will also pursue more practical settings for CGL.

Next, we have also made substantially technical contribution. Our proposed Hierarchical Prototype Networks (HPNs) continuously extract different levels of abstract knowledge (in the form of prototypes) from streams of tasks on graph representation learning. The continual learning performance of HPNs is both theoretically analyzed and experimentally justified from different perspectives on multiple public datasets with up to millions of nodes and tens of millions of edges. Besides, the memory consumption of HPNs is also theoretically analyzed and experimentally verified. HPNs provide a general framework for continual graph representation learning and can be further extended to tackle more challenging tasks. In the

future, we will extend HPNs to heterogeneous graphs by adapting the AFEs and prototypes to capture multiple types of nodes and edges. Moreover, we will also equip the prototypes with connections to denote the relationship among the abstract knowledge and investigate more challenging graph reasoning tasks.

Besides HPNs, we also propose two memory-replay based CGL techniques. The Subgraph Episodic Memory (SEM) stores the topological information in the form of sparsified computation subgraphs to perform continual graph representation learning. By sampling based graph sparsification, we reduce the memory consumption of a computation subgraph from $\mathcal{O}(d^L)$ to $\mathcal{O}(1)$, and for the first time enable GNNs to fully utilize the explicit topological information for memory replay. Our proposed method outperforms the existing state-of-the-art methods by a large margin on 4 public datasets in the class-IL (more practical and challenging) scenario.

Based on SSM, we further develop the Ricci curvature based graph sparsification technique to perform continual graph representation learning. We employed Subgraph Episodic Memory (SEM) to store the topological information in the form of computation subgraphs. By leveraging Ricci curvature to assess the informativeness, we can further enable GNNs to utilize the most informative topological information for memory replay. Our proposed method outperforms the existing state-of-the-art methods on 4 public datasets in both class-IL (more practical and challenging) and task-IL scenarios.

7.2 Future Research

7.2.1 Benchmark perspective

Continual graph learning is a newly emerging area, and inevitably more datasets, benchmarks, and evaluation tools will be needed to facilitate its development. Therefore, we will keep adding new contents to CGLB including new benchmark tasks constructed from other data sources, new benchmark results obtained from new methods, as well as an improved toolkit for the possible new evaluation requirements. All materials are accessible through https://github.com/QueuQ/CGLB.

Currently, CGLB only includes homogeneous graph datasets for NCGL tasks. In the future, we will also include benchmarks on heterogeneous graphs. First, we are now constructing benchmark tasks from several widely adopted data sources for heterogeneous graphs, including DBLP (academic network), IMDB (film rating network), Yelp (social media network), Amazon (E-commercial network). IMDB and Amazon are obtained via https://grouplens.org/datasets/movielens/100k/ and http://jmcauley.ucsd.edu/data/amazon/. While the curation of DBLP and Yelp follows the process proposed in a heterogeneous graph benchmark work [108]. The task construction will follow the node-level continual graph learning (NCGL) task construction proposed in our paper, with both task-IL and class-IL scenarios. Second, to implement the continual learning baselines on heterogeneous graphs, heterogeneous GNNs will have to be implemented as the backbones. Several representative heterogeneous GNNs including Heterogeneous Graph Neural Network (HetGNN) [119], Heterogeneous Graph Attention Network (HAN) [94], Heterogeneous Graph Transformer (HGT) [40], Relational Graph Convolutional Networks (R-GCNs) [78], Metapath2vec [25], and Predictive Text Embedding (PTE) [83] will be implemented into our continual learning pipelines, which will be benchmarked with the constructed tasks.

We will also keep exploring new learning scenarios. For example, continual learning on evolving graphs, a straightforward extension of our current work, is one of our future works. Specifically, an evolving graph may experience constant changes in both nodes and edges over time, and a model that constantly adapts to the newly incoming patterns may also experience forgetting the patterns learned from past periods. Therefore, the observational data of an evolving graph may be split into multiple periods, each of which corresponds to one task. Moreover, when the temporal boundaries are not clear, the setting will become task-free [4]. Real-world data exhibiting such behaviors include the financial data, traffic data, *etc.*. Besides, domain-incremental learning (domain-IL) [88, 122] can also be directly obtained by replacing the task-specific output heads with one head shared by all tasks and will be included in our benchmark. Specifically, an important aspect of constructing domain-IL tasks is to find the proper datasets in which data from different domains share the same label space. For example,

proteins from different human tissues [35] or different species [38] can be viewed as data from different domains, but their functions (labels) are in the same space.

7.2.2 Technical perspective

Although our proposed method achieves impressive performance on multiple datasets under different learning scenarios, there are still several perspectives that can be considered to further improve the model.

First, our proposed SSM and SEM require additional space for storing the sparsified computation subgraphs. Although we have demonstrated that the space consumption is acceptable on graphs up to millions of nodes and tens of millions of edges, some real-world graphs may be even larger and contain billions or even trillions of nodes. On such graphs, if the node attributes are highly diverse, maintaining a satisfying performance may require large memory buffers.

Second, with some databases, due to privacy reasons, storing information from the original graph may not be allowed. In this case, we may have to design additional modules to erase the identifying information before buffering the data.

Third, contrastive learning has achieved significant successes recently. For instance, Hypergraph Contrastive Collaborative Filtering [103] boosts the representation quality for recommender system based on hypergraph structure encoding and contrastive learning. Directed Graph Contrastive Learning [85] proposes to generate different views by perturbing the Laplacian matrix, so that the directed graph structure is not changed. The contrastive learning techniques is also promising for boosting the continual learning performance. Specifically, contrastive learning could avoid the dependency on the labels, which could significantly reduce the overfitting to each task and encourage the transferability of the learnt knowledge. In this way, the forgetting problem can also be alleviated. Towards this target, we will focus on two challenges.

- Although with less dependency on the task specific information, *i.e.* labels, contrastive learning based models still have to continually adapt to new distributions, and experience the forgetting problem. Moreover, in the class-IL setting, the final classifier, whose parameters are shared across different tasks, still needs supervised training and is subject to catastrophic forgetting.
- The existing continual learning works adopt task configurations constructed for supervised learning, and how to properly configure the tasks for the self-supervised learning setting requires careful consideration. For instance, how to split a given dataset into supervised and self-supervised parts, whether the model is allowed to access the self-supervised data all the time, etc.

Fourth, in this thesis, we target MPNNs working on homophilious graphs. However, learning graphs with heterophily are also practically important and are attracting increasingly more attention recently. For example, [133] systematically studies the challenges of learning on heterophilous graphs and proposes several techniques for designing effective heterophilous GNNs. [21] proposes a new constraint, *i.e.* homophily unnoticeability, to ensure that the Graph Injection Attack (GIA) preserves the graph homophily, so that the attack of GIA cannot be easily counteracted simply by homophily recovery. Due to its practical importance, extending CGL research to heterophilous graphs also deserves investigation. To achieve this aim, we will focus on two challenges. First, current CGL techniques are typically designed for MPNNs based on the message passing strategy. Since simple message passing is not suitable for heterophilous graphs [133], the techniques like curvature based graph sparsification in SEM-curvature is not directly applicable to heterophilous graph learning. Therefore, how to adapt the framework for analyzing information propagation to heterophilous GNNs becomes a key challenge. Second, existing works on continual graph learning do not focus on heterophilous graphs, and the benchmark tasks constructed from heterophilous graphs are lacking. Therefore, developing heterophilous CGL benchmarks are also targeted by our future works.

Bibliography

[1] Kian Ahrabian, Yishi Xu, Yingxue Zhang, Jiapeng Wu, Yuening Wang and Mark Coates. 'Structure Aware Experience Replay for Incremental Learning in Graph-based Recommender Systems'. In: *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 2021, pp. 2832–2836.

[2] Xing Ai, Chengyu Sun, Zhihong Zhang and Edwin R Hancock. 'Two-level Graph Neural Network'. In: *IEEE Transactions on Neural Networks and Learning Systems* (2022).

[3] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach and Tinne Tuytelaars. 'Memory aware synapses: Learning what (not) to forget'. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 139–154.

[4] Rahaf Aljundi, Klaas Kelchtermans and Tinne Tuytelaars. 'Task-free continual learning'. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 11254–11263.

[5] Rahaf Aljundi, Min Lin, Baptiste Goujaud and Yoshua Bengio. 'Gradient based sample selection for online continual learning'. In: *Advances in Neural Information Processing Systems*. 2019, pp. 11816–11825.

[6] Davide Bacciu and Danilo Numeroso. 'Explaining Deep Graph Networks via Input Perturbation'. In: *IEEE Transactions on Neural Networks and Learning Systems* (2022).

[7] Hoda Bahonar, Abdolreza Mirzaei, Saeed Sadri and Richard C Wilson. 'Graph embedding using frequency filtering'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.2 (2019), pp. 473–484.

[8] Smriti Bhagat, Graham Cormode and S Muthukrishnan. 'Node Classification in Social Networks'. In: *Social Network Data Analytics*. Springer, 2011, pp. 115–148.

[9] K. Bhatia, K. Dahiya, H. Jain, P. Kar, A. Mittal, Y. Prabhu and M. Varma. *The extreme classification repository: Multi-label datasets and code*. 2016. URL: http://manikvarma.org/downloads/XC/XMLRepository.html.

[10] Filippo Maria Bianchi, Daniele Grattarola, Lorenzo Livi and Cesare Alippi. 'Hierarchical representation learning in graph neural networks with node decimation pooling'. In: *IEEE Transactions on Neural Networks and Learning Systems* (2020).

[11] Hongbo Bo, Ryan McConville, Jun Hong and Weiru Liu. 'Ego-graph replay based continual learning for misinformation engagement prediction'. In: *2022 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2022, pp. 01–08.

[12] Caitlin R Bowman, Takako Iwashita and Dagmar Zeithamova. 'Tracking prototype and exemplar representations in the brain across learning'. In: *Elife* 9 (2020), e59360.

[13] Lucas Caccia, Eugene Belilovsky, Massimo Caccia and Joelle Pineau. 'Online learned continual compression with adaptive quantization modules'. In: *International Conference on Machine Learning*. PMLR. 2020, pp. 1240–1250.

[14] Jie Cai, Xin Wang, Chaoyu Guan, Yateng Tang, Jin Xu, Bin Zhong and Wenwu Zhu. 'Multimodal Continual Graph Learning with Neural Architecture Search'. In: *Proceedings of the ACM Web Conference 2022*. 2022, pp. 1292–1300.

[15] Daniele Calandriello, Alessandro Lazaric, Ioannis Koutis and Michal Valko. 'Improved large-scale graph learning through ridge spectral sparsification'. In: *ICML*. PMLR. 2018, pp. 688–697.

[16] Antonio Carta, Andrea Cossu, Federico Errica and Davide Bacciu. 'Catastrophic Forgetting in Deep Graph Networks: A Graph Classification Benchmark'. In: *Frontiers in artificial intelligence* 5 (2022).

[17] Alireza Chakeri, Hamidreza Farhidzadeh and Lawrence O Hall. 'Spectral sparsification in spectral clustering'. In: *ICPR*. IEEE. 2016, pp. 2301–2306.

[18] Jie Chen, Tengfei Ma and Cao Xiao. 'Fastgcn: fast learning with graph convolutional networks via importance sampling'. In: *arXiv preprint arXiv:1801.10247* (2018).

[19] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding and Yaliang Li. 'Simple and deep graph convolutional networks'. In: *International Conference on Machine Learning*. PMLR. 2020, pp. 1725–1735.

[20] Vincent S Chen, Paroma Varma, Ranjay Krishna, Michael Bernstein, Christopher Re and Li Fei-Fei. 'Scene graph prediction with limited labels'. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 2580–2590.

[21] Yongqiang Chen, Han Yang, Yonggang Zhang, Kaili Ma, Tongliang Liu, Bo Han and James Cheng. 'Understanding and improving graph injection attack by promoting unnoticeability'. In: *arXiv preprint arXiv:2202.08057* (2022).

[22] Aristotelis Chrysakis and Marie-Francine Moens. 'Online continual learning from imbalanced data'. In: *International Conference on Machine Learning*. PMLR. 2020, pp. 1952–1961.

[23] Tong Dandan, Zhu Haixue, Li Wenfu, Yang Wenjing, Qiu Jiang and Zhang Qinglin. 'Brain activity in using heuristic prototype to solve insightful problems'. In: *Behavioural Brain Research* 253 (2013), pp. 139–144.

[24] Angel Daruna, Mehul Gupta, Mohan Sridharan and Sonia Chernova. 'Continual learning of knowledge graph embeddings'. In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 1128–1135.

[25] Yuxiao Dong, Nitesh V Chawla and Ananthram Swami. 'metapath2vec: Scalable representation learning for heterogeneous networks'. In: *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*. 2017, pp. 135–144.

[26] Sefika Efeoglu. 'A Continual Relation Extraction Approach for Knowledge Graph Completeness (short paper)'. In: (2022).

[27] Mehrdad Farajtabar, Navid Azizan, Alex Mott and Ang Li. 'Orthogonal gradient descent for continual learning'. In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2020, pp. 3762–3773.

[28] Robin Forman. 'Bochner's method for cell complexes and combinatorial Ricci curvature'. In: *Discrete and Computational Geometry* 29.3 (2003), pp. 323–374.

[29] Lukas Galke, Iacopo Vagliano and Ansgar Scherp. 'Incremental training of graph neural networks on temporal graphs under distribution shift'. In: *arXiv preprint arXiv:2006.14422* (2020).

[30] Hongyang Gao, Yi Liu and Shuiwang Ji. 'Topology-aware graph pooling networks'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021).

[31] Spyros Gidaris and Nikos Komodakis. 'Dynamic few-shot visual learning without forgetting'. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 4367–4375.

[32] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals and George E Dahl. 'Neural message passing for quantum chemistry'. In: *International Conference on Machine Learning*. PMLR. 2017, pp. 1263–1272.

[33] Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville and Yoshua Bengio. 'An empirical investigation of catastrophic forgetting in gradient-based neural networks'. In: *arXiv preprint arXiv:1312.6211* (2013).

[34] Zhichun Guo, Chuxu Zhang, Wenhao Yu, John Herr, Olaf Wiest, Meng Jiang and Nitesh V Chawla. 'Few-shot graph learning for molecular property prediction'. In: *Proceedings of the Web Conference 2021*. 2021, pp. 2559–2567.

[35] Will Hamilton, Zhitao Ying and Jure Leskovec. 'Inductive representation learning on large graphs'. In: *Advances in Neural Information Processing Systems*. 2017, pp. 1024–1034.

[36] Yi Han, Shanika Karunasekera and Christopher Leckie. 'Graph neural networks with continual learning for fake news detection from social media'. In: *arXiv preprint arXiv:2007.03316* (2020).

[37] Yen-Chang Hsu, Yen-Cheng Liu, Anita Ramasamy and Zsolt Kira. 'Re-evaluating continual learning scenarios: A categorization and case for strong baselines'. In: *arXiv preprint arXiv:1810.12488* (2018).

[38] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta and Jure Leskovec. 'Open Graph Benchmark: Datasets for Machine Learning on Graphs'. In: *arXiv preprint arXiv:2005.00687* (2020).

[39] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta and Jure Leskovec. 'Open graph benchmark: Datasets for machine learning on graphs'. In: *Advances in neural information processing systems* 33 (2020), pp. 22118–22133.

[40] Ziniu Hu, Yuxiao Dong, Kuansan Wang and Yizhou Sun. 'Heterogeneous graph transformer'. In: *Proceedings of The Web Conference 2020*. 2020, pp. 2704–2710.

[41] Christian Hübler, Hans-Peter Kriegel, Karsten Borgwardt and Zoubin Ghahramani. 'Metropolis algorithms for representative subgraph sampling'. In: *ICDM*. IEEE. 2008, pp. 283–292.

[42] Heechul Jung, Jeongwoo Ju, Minju Jung and Junmo Kim. 'Less-forgetting learning in deep neural networks'. In: *arXiv preprint arXiv:1607.00122* (2016).

[43] Thomas N Kipf and Max Welling. 'Semi-supervised classification with graph convolutional networks'. In: *arXiv preprint arXiv:1609.02907* (2016).

[44] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska et al. 'Overcoming catastrophic forgetting in neural networks'. In: *Proceedings of the National Academy of Sciences* 114.13 (2017), pp. 3521–3526.

[45] Jeremias Knoblauch, Hisham Husain and Tom Diethe. 'Optimal continual learning has perfect memory and is np-hard'. In: *International Conference on Machine Learning*. PMLR. 2020, pp. 5327–5337.

[46] Xiaoyu Kou, Yankai Lin, Shaobo Liu, Peng Li, Jie Zhou and Yan Zhang. 'Disentangle-based Continual Graph Representation Learning'. In: *arXiv preprint arXiv:2010.02565* (2020).

[47] Alex Krizhevsky, Geoffrey Hinton et al. 'Learning multiple layers of features from tiny images'. In: (2009).

[48] Dharshan Kumaran, Demis Hassabis and James L McClelland. 'What learning systems do intelligent agents need? Complementary learning systems theory updated'. In: *Trends in Cognitive Sciences* 20.7 (2016), pp. 512–534.

[49] Huan Lei, Naveed Akhtar and Ajmal Mian. 'Spherical kernel for efficient graph convolution on 3d point clouds'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2020).

[50] Jure Leskovec and Christos Faloutsos. 'Sampling from large graphs'. In: *KDD*. 2006, pp. 631–636.

[51] Guohao Li, Matthias Muller, Ali Thabet and Bernard Ghanem. 'Deepgcns: Can gcns go as deep as cnns?' In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 9267–9276.

[52] Zhizhong Li and Derek Hoiem. 'Learning without forgetting'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40.12 (2017), pp. 2935–2947.

[53] David Liben-Nowell and Jon Kleinberg. 'The link-prediction problem for social networks'. In: *Journal of the American Society for Information Science and Technology* 58.7 (2007), pp. 1019–1031.

[54] Wanyu Lin, Hao Lan and Baochun Li. 'Generative causal explanations for graph neural networks'. In: *ICML*. PMLR. 2021, pp. 6666–6679.

[55] Yong Lin, Linyuan Lu and Shing-Tung Yau. 'Ricci curvature of graphs'. In: *Tohoku Mathematical Journal, Second Series* 63.4 (2011), pp. 605–627.

[56] Zhiqiu Lin, Jia Shi, Deepak Pathak and Deva Ramanan. 'The CLEAR Benchmark: Continual LEArning on Real-World Imagery'. In: *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*. 2021.

[57] Huihui Liu, Yiding Yang and Xinchao Wang. 'Overcoming catastrophic forgetting in graph neural networks'. In: *arXiv preprint arXiv:2012.06002* (2020).

[58] Huihui Liu, Yiding Yang and Xinchao Wang. 'Overcoming catastrophic forgetting in graph neural networks'. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 10. 2021, pp. 8653–8661.

[59] David Lopez-Paz and Marc'Aurelio Ranzato. 'Gradient episodic memory for continual learning'. In: *Advances in Neural Information Processing Systems*. 2017, pp. 6467–6476.

[60] Dongsheng Luo, Wei Cheng, Dongkuan Xu, Wenchao Yu, Bo Zong, Haifeng Chen and Xiang Zhang. 'Parameterized explainer for graph neural network'. In: *NeurIPS* 33 (2020), pp. 19620–19631.

[61] Yao Ma, Ziyi Guo, Zhaocun Ren, Jiliang Tang and Dawei Yin. 'Streaming graph neural networks'. In: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2020, pp. 719–728.

[62] Laurens Van der Maaten and Geoffrey Hinton. 'Visualizing data using t-SNE.' In: *Journal of Machine Learning Research* 9.11 (2008).

[63] Arun S Maiya and Tanya Y Berger-Wolf. 'Sampling community structure'. In: *Proceedings of the 19th international conference on World wide web*. 2010, pp. 701–710.

[64] Michael Mathioudakis, Francesco Bonchi, Carlos Castillo, Aristides Gionis and Antti Ukkonen. 'Sparsification of influence networks'. In: *KDD*. 2011, pp. 529–537.

[65] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie and Kristie Seymore. 'Automating the construction of internet portals with machine learning'. In: *Information Retrieval* 3.2 (2000), pp. 127–163.

[66] James L McClelland, Bruce L McNaughton and Randall C O'Reilly. 'Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory.' In: *Psychological Review* 102.3 (1995), p. 419.

[67] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado and Jeffrey Dean. 'Distributed representations of words and phrases and their compositionality'. In: *arXiv preprint arXiv:1310.4546* (2013).

[68] Giang Hoang Nguyen, John Boaz Lee, Ryan A Rossi, Nesreen K Ahmed, Eunyee Koh and Sungchul Kim. 'Continuous-time dynamic network embeddings'. In: *Companion Proceedings of the The Web Conference 2018*. 2018, pp. 969–976.

[69] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei and Bo Yang. 'Geom-gcn: Geometric graph convolutional networks'. In: *arXiv preprint arXiv:2002.05287* (2020).

[70] Massimo Perini, Giorgia Ramponi, Paris Carbone and Vasiliki Kalavri. 'Learning on streaming graphs with experience replay'. In: *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*. 2022, pp. 470–478.

[71] Bryan Perozzi, Rami Al-Rfou and Steven Skiena. 'Deepwalk: Online learning of social representations'. In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2014, pp. 701–710.

[72] Ameya Prabhu, Philip HS Torr and Puneet K Dokania. 'Gdumb: A simple approach that questions our progress in continual learning'. In: *European conference on computer vision*. Springer. 2020, pp. 524–540.

[73] Appan Rakaraddi, Lam Siew Kei, Mahardhika Pratama and Marcus De Carvalho. 'Reinforced Continual Learning for Graphs'. In: *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*. 2022, pp. 1666–1674.

[74] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl and Christoph H Lampert. 'icarl: Incremental classifier and representation learning'. In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2017, pp. 2001–2010.

[75] Yu Rong, Wenbing Huang, Tingyang Xu and Junzhou Huang. 'Dropedge: Towards deep graph convolutional networks on node classification'. In: *International Conference on Learning Representations*. 2019.

[76] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu and Raia Hadsell. 'Progressive neural networks'. In: *arXiv preprint arXiv:1606.04671* (2016).

[77] Gobinda Saha and Kaushik Roy. 'Gradient projection memory for continual learning'. In: *International Conference on Learning Representation*. 2021.

[78] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov and Max Welling. 'Modeling relational data with graph convolutional networks'. In: *European semantic web conference*. Springer. 2018, pp. 593–607.

[79] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher and Tina Eliassi-Rad. 'Collective classification in network data'. In: *AI Magazine* 29.3 (2008), pp. 93–93.

[80] Hanul Shin, Jung Kwon Lee, Jaehong Kim and Jiwon Kim. 'Continual learning with deep generative replay'. In: *Advances in Neural Information Processing Systems*. 2017, pp. 2990–2999.

[81] Indro Spinelli, Simone Scardapane and Aurelio Uncini. 'A Meta-Learning Approach for Training Explainable Graph Neural Networks'. In: *IEEE Transactions on Neural Networks and Learning Systems* (2022).

[82] RP Sreejith, Karthikeyan Mohanraj, Jürgen Jost, Emil Saucan and Areejit Samal. 'Forman curvature for complex networks'. In: *Journal of Statistical Mechanics: Theory and Experiment* 2016.6 (2016), p. 063206.

[83] Jian Tang, Meng Qu and Qiaozhu Mei. 'Pte: Predictive text embedding through large-scale heterogeneous text networks'. In: *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. 2015, pp. 1165–1174.

[84] Jie Tang, Jimeng Sun, Chi Wang and Zi Yang. 'Social influence analysis in large-scale networks'. In: *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge discovery and data mining*. 2009, pp. 807–816.

[85] Zekun Tong, Yuxuan Liang, Henghui Ding, Yongxing Dai, Xinke Li and Changhu Wang. 'Directed graph contrastive learning'. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 19580–19593.

[86] Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong and Michael M Bronstein. 'Understanding over-squashing and bottlenecks on graphs via curvature'. In: *arXiv preprint arXiv:2111.14522* (2021).

[87] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio and Yoshua Bengio. 'Graph attention networks'. In: *arXiv preprint arXiv:1710.10903* (2017).

[88] Gido M Van de Ven and Andreas S Tolias. 'Three scenarios for continual learning'. In: *arXiv preprint arXiv:1904.07734* (2019).

[89] Chen Wang, Yuheng Qiu and Sebastian Scherer. 'Bridging graph network to lifelong learning with feature interaction'. In: (2020).

[90] Chen Wang, Yuheng Qiu and Sebastian Scherer. 'Lifelong graph learning'. In: *arXiv preprint arXiv:2009.00647* (2020).

[91] Hao Wang, Yitong Wang, Zheng Zhou, Xing Ji, Dihong Gong, Jingchao Zhou, Zhifeng Li and Wei Liu. 'Cosface: Large margin cosine loss for deep face recognition'. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 5265–5274.

[92] Junshan Wang, Guojie Song, Yi Wu and Liang Wang. 'Streaming graph neural networks via continual learning'. In: *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 2020, pp. 1515–1524.

[93] Kuansan Wang, Zhihong Shen, Chiyuan Huang, Chieh-Han Wu, Yuxiao Dong and Anshul Kanakia. 'Microsoft academic graph: When experts are not enough'. In: *Quantitative Science Studies* 1.1 (2020), pp. 396–413.

[94] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui and Philip S Yu. 'Heterogeneous graph attention network'. In: *The world wide web conference*. 2019, pp. 2022–2032.

[95] Zhen Wang, Zhaoqing Li, Rong Wang, Feiping Nie and Xuelong Li. 'Large graph clustering with simultaneous spectral embedding and discretization'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2020).

[96] Mitchell Wortsman, Vivek Ramanujan, Rosanne Liu, Aniruddha Kembhavi, Mohammad Rastegari, Jason Yosinski and Ali Farhadi. 'Supermasks in superposition'. In: *arXiv preprint arXiv:2006.14769* (2020).

[97] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu and Kilian Weinberger. 'Simplifying graph convolutional networks'. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 6861–6871.

[98] Guile Wu, Shaogang Gong and Pan Li. 'Striking a balance between stability and plasticity for class-incremental learning'. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 1124–1133.

[99] Xinyu Wu, Ye Yuan, Xikun Zhang, Can Wang, Tiantian Xu and Dacheng Tao. 'Gait phase classification for a lower limb exoskeleton system based on a graph convolutional network model'. In: *IEEE Transactions on Industrial Electronics* 69.5 (2021), pp. 4999–5008.

[100] Yue Wu, Yinpeng Chen, Lijuan Wang, Yuancheng Ye, Zicheng Liu, Yandong Guo and Yun Fu. 'Large scale incremental learning'. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 374–382.

[101] Zhenqin Wu, Bharath Ramsundar, Evan N Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S Pappu, Karl Leswing and Vijay Pande. 'MoleculeNet: a benchmark for molecular machine learning'. In: *Chemical science* 9.2 (2018), pp. 513–530.

[102] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang and S Yu Philip. 'A comprehensive survey on graph neural networks'. In: *IEEE Transactions on Neural Networks and Learning Systems* 32.1 (2020), pp. 4–24.

[103] Lianghao Xia, Chao Huang, Yong Xu, Jiashu Zhao, Dawei Yin and Jimmy Huang. 'Hypergraph contrastive collaborative filtering'. In: *Proceedings of the 45th International ACM SIGIR conference on research and development in information retrieval*. 2022, pp. 70–79.

[104] Zhaoping Xiong, Dingyan Wang, Xiaohong Liu, Feisheng Zhong, Xiaozhe Wan, Xutong Li, Zhaojun Li, Xiaomin Luo, Kaixian Chen, Hualiang Jiang et al. 'Pushing the boundaries of molecular representation for drug discovery with the graph attention mechanism'. In: *Journal of medicinal chemistry* 63.16 (2019), pp. 8749–8760.

[105] Keyulu Xu, Weihua Hu, Jure Leskovec and Stefanie Jegelka. 'How powerful are graph neural networks?' In: *arXiv preprint arXiv:1810.00826* (2018).

[106] Yishi Xu, Yingxue Zhang, Wei Guo, Huifeng Guo, Ruiming Tang and Mark Coates. 'Graphsail: Graph structure aware incremental learning for recommender systems'. In: *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 2020, pp. 2861–2868.

[107] Yufei Xu, Qiming Zhang, Jing Zhang and Dacheng Tao. 'Vitae: Vision transformer advanced by exploring intrinsic inductive bias'. In: *Advances in Neural Information Processing Systems* 34 (2021).

[108] Carl Yang, Yuxin Xiao, Yu Zhang, Yizhou Sun and Jiawei Han. 'Heterogeneous network representation learning: A unified framework with survey and benchmark'. In: *IEEE Transactions on Knowledge and Data Engineering* (2020).

[109] Huaxiu Yao, Chuxu Zhang, Ying Wei, Meng Jiang, Suhang Wang, Junzhou Huang, Nitesh Chawla and Zhenhui Li. 'Graph few-shot learning via knowledge transfer'. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 04. 2020, pp. 6656–6663.

[110] Ze Ye, Kin Sum Liu, Tengfei Ma, Jie Gao and Chao Chen. 'Curvature graph network'. In: *ICLR*. 2019.

[111] Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik and Jure Leskovec. 'Gnnexplainer: Generating explanations for graph neural networks'. In: *NeurIPS* 32 (2019).

[112] Jaehong Yoon, Saehoon Kim, Eunho Yang and Sung Ju Hwang. 'Scalable and order-robust continual learning with additive parameter decomposition'. In: *International Conference on Learning Representation*. 2020.

[113] Jaehong Yoon, Eunho Yang, Jeongtae Lee and Sung Ju Hwang. 'Lifelong learning with dynamically expandable networks'. In: *arXiv preprint arXiv:1708.01547* (2017).

[114] Junchi Yu, Tingyang Xu, Yu Rong, Yatao Bian, Junzhou Huang and Ran He. 'Graph information bottleneck for subgraph recognition'. In: *arXiv preprint arXiv:2010.05563* (2020).

[115] Wenchao Yu, Wei Cheng, Charu C Aggarwal, Kai Zhang, Haifeng Chen and Wei Wang. 'Netwalk: A flexible deep embedding approach for anomaly detection in dynamic networks'. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2018, pp. 2672–2681.

[116] Anam Zaman, Fan Yangyu, Muhammad Saad Ayub, Muhammad Irfan, Lv Guoyun and Liu Shiya. 'CMDGAT: knowledge extraction and retention based continual graph attention network for point cloud registration'. In: *Expert Systems with Applications* (2022), p. 119098.

[117] Dagmar Zeithamova, W Todd Maddox and David M Schnyer. 'Dissociable prototype learning systems: Evidence from brain imaging and behavior'. In: *Journal of Neuroscience* 28.49 (2008), pp. 13194–13201.

[118] Friedemann Zenke, Ben Poole and Surya Ganguli. 'Continual learning through synaptic intelligence'. In: *International Conference on Machine Learning*. PMLR. 2017, pp. 3987–3995.

[119] Chuxu Zhang, Dongjin Song, Chao Huang, Ananthram Swami and Nitesh V Chawla. 'Heterogeneous graph neural network'. In: *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2019, pp. 793–803.

[120] Qiming Zhang, Yufei Xu, Jing Zhang and Dacheng Tao. 'Vitaev2: Vision transformer advanced by exploring inductive bias for image recognition and beyond'. In: *arXiv preprint arXiv:2202.10108* (2022).

[121] Xikun Zhang, Dongjin Song and Dacheng Tao. 'Cglb: Benchmark tasks for continual graph learning'. In: *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*. 2022.

[122] Xikun Zhang, Dongjin Song and Dacheng Tao. 'Hierarchical Prototype Networks for Continual Graph Representation Learning'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2022).

[123] Xikun Zhang, Dongjin Song and Dacheng Tao. 'Ricci Curvature-Based Graph Sparsification for Continual Graph Representation Learning'. In: *IEEE Transactions on Neural Networks and Learning Systems* (2023).

[124] Xikun Zhang, Dongjin Song and Dacheng Tao. 'Sparsified Subgraph Memory for Continual Graph Representation LearningGated Information Bottleneck for Generalization in Sequential Environments'. In: *2022 IEEE International Conference on Data Mining (ICDM)*. IEEE. 2022.

[125] Xikun Zhang, Chang Xu and Dacheng Tao. 'Context aware graph convolution for skeleton-based action recognition'. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 14333–14342.

[126] Xikun Zhang, Chang Xu and Dacheng Tao. 'On dropping clusters to regularize graph convolutional neural networks'. In: *European Conference on Computer Vision*. 2020.

[127] Xikun Zhang, Chang Xu, Xinmei Tian and Dacheng Tao. 'Graph edge convolutional neural networks for skeleton-based action recognition'. In: *IEEE Transactions on Neural Networks and Learning Systems* 31.8 (2019), pp. 3047–3060.

[128] Cheng Zheng, Bo Zong, Wei Cheng, Dongjin Song, Jingchao Ni, Wenchao Yu, Haifeng Chen and Wei Wang. 'Robust graph representation learning via neural sparsification'. In: *ICML*. PMLR. 2020, pp. 11458–11468.

[129] Fan Zhou and Chengtai Cao. 'Overcoming catastrophic forgetting in graph neural networks with experience replay'. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 5. 2021, pp. 4714–4722.

[130] Fan Zhou, Chengtai Cao, Kunpeng Zhang, Goce Trajcevski, Ting Zhong and Ji Geng. 'Meta-gnn: On few-shot node classification in graph meta-learning'. In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 2019, pp. 2357–2360.

[131] Fan Zhou, Chengtai Cao, Ting Zhong, Kunpeng Zhang, Goce Trajcevski and Ji Geng. 'Continual graph learning'. In: *arXiv preprint arXiv:2003.09908* (2020).

[132] Lekui Zhou, Yang Yang, Xiang Ren, Fei Wu and Yueting Zhuang. 'Dynamic network embedding by modeling triadic closure process'. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018.

[133] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu and Danai Koutra. 'Beyond homophily in graph neural networks: Current limitations and effective designs'. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 7793–7804.

[134] Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun and Quanquan Gu. 'Layer-Dependent importance sampling for training deep and large graph convolutional networks'. In: *Advances in Neural Information Processing Systems*. 2019, pp. 11247–11256.