Sparsified Subgraph Memory for Continual Graph Representation Learning

Xikun Zhang
The University of Sydney, Australia
xzha0505@uni.sydney.edu.au

Dongjin Song
University of Connecticut, USA
dongjin.song@uconn.edu

Dacheng Tao
The University of Sydney, Australia
dacheng.tao@gmail.com

***Abstract*—Memory replay, which stores a subset of representative historical data from previous tasks to replay while learning new tasks, exhibits state-of-the-art performance for various continual learning applications on Euclidean data. While topological information plays a critical role in characterizing graph data, existing memory replay based graph learning techniques only store individual nodes for replay and do not consider their associated edge information. To this end, we propose a sparsified subgraph memory (SSM), which sparsifies the selected computation graphs into fixed size before storing them into the memory. In this way, we can 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 utilize the explicit topological information for memory replay. Finally, our empirical studies show that SSM outperforms state-of-the-art approaches by up to 27.8% on four different public datasets. Unlike existing methods which focus on task incremental learning (task-IL) setting, SSM succeeds in the challenging class incremental learning (class-IL) setting in which a model is required to distinguish all learned classes without task indicators, and even achieves comparable performance to joint training which is the performance upper bound for continual learning. Our code is available at https://github.com/QueuQ/SSM.**

***Index Terms*—Graph representation learning, continual learning, graph sparsification.**

I. Introduction

In real-world graph applications, it is critical to ensure that Graph Neural Networks (GNNs) [1]–[3] 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 [4], [5], 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 [6]–[8]. Due to its success, memory replay is also adopted for continual learning on graph data by storing and replaying representative nodes [9]. 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 paper, 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 [2], 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 [10]–[17], however, most of them are not applicable for continual graph representation learning as detailed in Section II-B.

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.

2374-8486/22/$31.00 ©2022 IEEE
DOI 10.1109/ICDM54844.2022.00177

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.

II. Related Works

Our work is closely related to continual learning, continual graph learning, and graph sparsification.

A. Continual Learning & Continual Graph Learning

Machine learning models often encounter the catastrophic forgetting problem when adapting to new tasks. To resolve this challenge, existing approaches are roughly divided into three categories. Regularization based methods prevent drastic modification to model parameters important to previous tasks via different constraints [18]–[21]. Parametric isolation based methods protect the important parameters by allocating new parameters for new tasks [22]–[25]. Finally, memory replay based methods alleviate forgetting by replaying representative data from previous tasks when learning new tasks [6]–[8].

Recently, continual learning on graph data is also attracting increasingly more attention due to its value in practical scenarios. Several methods and benchmarks have been developed [9], [26]–[28]. These methods include regularization based ones like topology-aware weight preserving (TWP) [27] which preserves crucial parameters and topologies via regularization; parametric isolation based approaches like HPNs [26] which adaptively select different parameter combinations for different tasks; and memory replay based methods like ER-GNN [9] which stores representative nodes which are replayed when learning new tasks. Our proposed work is also based on memory replay, and the key advantage of our model is that we store explicit topological information with a manageable space complexity which has never been achieved before.

Finally, it is also worth noting the difference between continual graph representation learning and other settings. Dynamic graph representation learning [29]–[34] focuses on capturing the temporal node dynamics with all previous information being accessible. On the contrary, continual graph representation learning focuses on alleviating the forgetting on previous tasks, therefore the previous data is inaccessible when learning new tasks. Few-shot graph learning [35], [36] aims at fast adapting the model to new tasks. During training, few-shot learning models can access to data of all tasks simultaneously, while models under the continual learning setting can only access the data of the current task. 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 on existing tasks without any fine-tuning.

B. Graph Sparsification

Graph sparsification has been extensively studied in the past few years [10]–[17]. However, unlike the strategies adopted 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 [10]–[14], [37]. In addition, these approaches may also encounter high computational burdens. Recently, various GNN explanation [15], [16], [38], [39] or graph denoising [17] 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 [3] utilizes randomly sampled neighbors at each layer to reduce the computational burden. In this work, we adopt two sparsification strategies. One is based on uniform sampling, which is similar to the one used in GraphSAGE but without replacement. Another is an importance sampling that samples the neighbors iteratively based on the node degree distribution.

III. 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.

A. 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 $u \in \mathbb{V}_i$:

$$\mathcal{N}^L(u) = \{v \in \mathbb{V}_\tau | \mathrm{d}(u, v) = L\}, \tag{1}$$

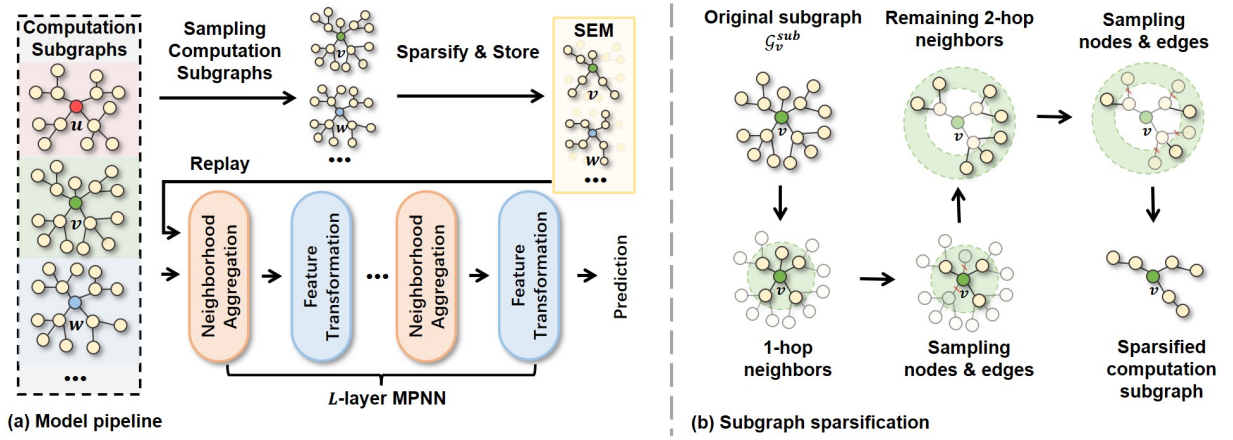

Fig. 1. (a) The pipeline of SEM. (b) The iterative sparsification process for a computation subgraph containing 2-hop neighbors.

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 [2]. In this work, we focus on the MPNNs and refer all GNNs to MPNNs in the following.

B. 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:

$$\mathcal{L} = \underbrace{\sum_{\mathbf{x}_i \in \mathbb{D}_\tau} l(f(\mathbf{x}_i;\boldsymbol{\theta}), \mathbf{y}_i)}_{\text{loss of the current task } \mathcal{L}_\tau} + \lambda \underbrace{\sum_{\mathbf{x}_j \in \mathcal{B}} l(f(\mathbf{x}_j;\boldsymbol{\theta}), \mathbf{y}_j)}_{\text{auxiliary loss } \mathcal{L}_{aux}}, \quad (2)$$

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 [7] 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 [6] 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:

$$\mathbf{m}_u^{l+1} = \sum_{v \in \mathcal{N}^1(u)} \mathrm{M}_l(\mathbf{h}_u^l, \mathbf{h}_v^l, \mathbf{x}_{u,v}^e; \boldsymbol{\theta}_l^{\mathrm{M}}), \quad (3)$$

$$\mathbf{h}_u^{l+1} = \mathrm{U}_l(\mathbf{h}_u^l, \mathbf{m}_u^{l+1}; \boldsymbol{\theta}_l^{\mathrm{U}}), \quad (4)$$

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:

$$\mathbf{h}_u^L = \mathrm{MPNN}(\mathcal{G}_u^{sub}; \boldsymbol{\Theta}), \quad (5)$$

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.

Algorithm 1 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 $\mathbb{E}$.

C. 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 1.

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:

$$\mathcal{L} = \underbrace{\sum_{u \in \mathbb{V}_\tau} l(\text{MPNN}(\mathcal{G}_u^{sub}; \mathbf{\Theta}), \mathbf{y}_u)}_{\text{loss of the current task } \mathcal{L}_\tau} + \lambda \underbrace{\sum_{\bar{\mathcal{G}}_v^{sub} \in \mathcal{SEM}} l(\text{MPNN}(\bar{\mathcal{G}}_v^{sub}; \mathbf{\Theta}), \mathbf{y}_v)}_{\text{auxiliary loss } \mathcal{L}_{aux}}. \quad (6)$$

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 IV-B3.

TABLE I
STATISTICS OF DATASETS AND TASK SPLITTINGS

Dataset	CoraFull [40]	OGB-Arxiv [1]	Reddit [3]	OGB-Products [2]
# 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

IV. 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?

A. Datasets

We followed the Continual Graph Learning Benchmark (CGLB) [28] 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 I.

B. Experimental Settings

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 [28]. 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*.

For continual learning, the most thorough evaluation is the accuracy matrix $\mathrm{M}^{acc} \in \mathbb{R}^{T \times T}$. Each entry $\mathrm{M}_{i,j}^{acc}$ denotes the model's accuracy on task j after learning task i. Each row $\mathrm{M}_{i,:}^{acc}$ shows the accuracy on all previous tasks after learning task i. Each column $\mathrm{M}_{:,j}^{acc}$ 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) $\frac{\sum_{i=1}^{T} \mathrm{M}_{T,i}^{acc}}{T}$ and average forgetting (AF)

[1] https://ogb.stanford.edu/docs/nodeprop/#ogbn-arxiv
[2] https://ogb.stanford.edu/docs/nodeprop/#ogbn-products

TABLE II
PERFORMANCE COMPARISONS UNDER CLASS-IL ON 4 DATASETS (↑ HIGHER MEANS BETTER).

Continual learning 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 [19]	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 [21]	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 [7]	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 [27]	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 [20]	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 [9]	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 (Not under continual setting)	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
SEM-uniform (Ours)	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
SEM-degree (Ours)	**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

$\frac{\sum_{i=1}^{T-1} \mathrm{M}_{T,i}^{acc} - \mathrm{M}_{i,i}^{acc}}{T-1}$ 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.

2) Baselines and model settings: Our baselines are adopted from CGLB [28] including Experience Replay based GNN (ERGNN) [9], Topology-aware Weight Preserving (TWP) [27], Elastic Weight Consolidation (EWC) [19], Learning without Forgetting (LwF) [20], Gradient Episodic Memory (GEM) [7], and Memory Aware Synapses (MAS) [21]). We also adopt joint training as the upper bound [26]–[28]. 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 [26]–[28]. All models are implemented based on four popular backbone GNNs, *i.e.*, Graph Convolutional Networks (GCNs) [1], Simple Graph Convolution (SGC) [41], Graph Attention Networks (GATs) [42], and Graph Isomorphism Network (GIN) [43]. 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.

3) Class imbalance & class-IL classifier: According to Section IV-B1, 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:

$$\mathcal{L} = \sum_{v \in \mathbb{V}_\tau} l(f(\mathbf{e}_v; \boldsymbol{\theta}), y_v) \cdot s_{y_v} + \sum_{\mathbf{e}_w \in \mathcal{SEM}} l(f(\mathbf{e}_w; \boldsymbol{\theta}), y_w) \cdot s_{y_w}, \tag{7}$$

λ in Equation (6) 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 [44], [45] 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.

C. Comparisons for Class-IL Scenario (RQ1, 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 II, 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.

V. CONCLUSION

In this paper, 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.

ACKNOWLEDGMENT

Dacheng Tao and Xikun Zhang are supported by Australian Research Council Projects FL-170100117, DP-180103424, IH-180100002, and IC-190100031. Dongjin Song is supported by UConn Scholarship Facilitation Fund (SFF) Award, NSF CNS-2154191, and USDA NIFA 2022-67022-36603.

REFERENCES

[1] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[2] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *ICML*. PMLR, 2017, pp. 1263–1272.

[3] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NeurIPS*, 2017, pp. 1024–1034.

[4] J. L. McClelland, B. L. McNaughton, and R. 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." *Psychological Review*, vol. 102, no. 3, p. 419, 1995.

[5] D. Kumaran, D. Hassabis, and J. L. McClelland, "What learning systems do intelligent agents need? complementary learning systems theory updated," *Trends in Cognitive Sciences*, vol. 20, no. 7, pp. 512–534, 2016.

[6] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, "icarl: Incremental classifier and representation learning," in *CVPR*, 2017, pp. 2001–2010.

[7] D. Lopez-Paz and M. Ranzato, "Gradient episodic memory for continual learning," in *NeurIPS*, 2017, pp. 6467–6476.

[8] R. Aljundi, M. Lin, B. Goujaud, and Y. Bengio, "Gradient based sample selection for online continual learning," in *NeurIPS*, 2019, pp. 11 816–11 825.

[9] F. Zhou and C. Cao, "Overcoming catastrophic forgetting in graph neural networks with experience replay," in *AAAI*, vol. 35, no. 5, 2021, pp. 4714–4722.

[10] C. Hübler, H.-P. Kriegel, K. Borgwardt, and Z. Ghahramani, "Metropolis algorithms for representative subgraph sampling," in *ICDM*. IEEE, 2008, pp. 283–292.

[11] M. Mathioudakis, F. Bonchi, C. Castillo, A. Gionis, and A. Ukkonen, "Sparsification of influence networks," in *KDD*, 2011, pp. 529–537.

[12] D. Calandriello, A. Lazaric, I. Koutis, and M. Valko, "Improved large-scale graph learning through ridge spectral sparsification," in *ICML*. PMLR, 2018, pp. 688–697.

[13] A. Chakeri, H. Farhidzadeh, and L. O. Hall, "Spectral sparsification in spectral clustering," in *ICPR*. IEEE, 2016, pp. 2301–2306.

[14] J. Leskovec and C. Faloutsos, "Sampling from large graphs," in *KDD*, 2006, pp. 631–636.

[15] Z. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec, "Gnnexplainer: Generating explanations for graph neural networks," *NeurIPS*, vol. 32, 2019.

[16] D. Luo, W. Cheng, D. Xu, W. Yu, B. Zong, H. Chen, and X. Zhang, "Parameterized explainer for graph neural network," *NeurIPS*, vol. 33, pp. 19 620–19 631, 2020.

[17] C. Zheng, B. Zong, W. Cheng, D. Song, J. Ni, W. Yu, H. Chen, and W. Wang, "Robust graph representation learning via neural sparsification," in *ICML*. PMLR, 2020, pp. 11 458–11 468.

[18] M. Farajtabar, N. Azizan, A. Mott, and A. Li, "Orthogonal gradient descent for continual learning," in *ICAIS*. PMLR, 2020, pp. 3762–3773.

[19] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the National Academy of Sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[20] Z. Li and D. Hoiem, "Learning without forgetting," *TPAMI*, vol. 40, no. 12, pp. 2935–2947, 2017.

[21] R. Aljundi, F. Babiloni, M. Elhoseiny, M. Rohrbach, and T. Tuytelaars, "Memory aware synapses: Learning what (not) to forget," in *ECCV*, 2018, pp. 139–154.

[22] M. Wortsman, V. Ramanujan, R. Liu, A. Kembhavi, M. Rastegari, J. Yosinski, and A. Farhadi, "Supermasks in superposition," *arXiv preprint arXiv:2006.14769*, 2020.

[23] J. Yoon, S. Kim, E. Yang, and S. J. Hwang, "Scalable and order-robust continual learning with additive parameter decomposition," in *ICLR*, 2020.

[24] J. Yoon, E. Yang, J. Lee, and S. J. Hwang, "Lifelong learning with dynamically expandable networks," *arXiv preprint arXiv:1708.01547*, 2017.

[25] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive neural networks," *arXiv preprint arXiv:1606.04671*, 2016.

[26] X. Zhang, D. Song, and D. Tao, "Hierarchical prototype networks for continual graph representation learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[27] H. Liu, Y. Yang, and X. Wang, "Overcoming catastrophic forgetting in graph neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 10, 2021, pp. 8653–8661.

[28] X. Zhang, D. Song, and D. Tao, "Cglb: Benchmark tasks for continual graph learning," in *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022.

[29] L. Galke, I. Vagliano, and A. Scherp, "Incremental training of graph neural networks on temporal graphs under distribution shift," *arXiv preprint arXiv:2006.14422*, 2020.

[30] J. Wang, G. Song, Y. Wu, and L. Wang, "Streaming graph neural networks via continual learning," in *CIKM*, 2020, pp. 1515–1524.

[31] W. Yu, W. Cheng, C. C. Aggarwal, K. Zhang, H. Chen, and W. Wang, "Netwalk: A flexible deep embedding approach for anomaly detection in dynamic networks," in *KDD*, 2018, pp. 2672–2681.

[32] G. H. Nguyen, J. B. Lee, R. A. Rossi, N. K. Ahmed, E. Koh, and S. Kim, "Continuous-time dynamic network embeddings," in *Companion Proceedings of the The Web Conference 2018*, 2018, pp. 969–976.

[33] L. Zhou, Y. Yang, X. Ren, F. Wu, and Y. Zhuang, "Dynamic network embedding by modeling triadic closure process," in *AAAI*, vol. 32, no. 1, 2018.

[34] Y. Ma, Z. Guo, Z. Ren, J. Tang, and D. 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.

[35] Z. Guo, C. Zhang, W. Yu, J. Herr, O. Wiest, M. Jiang, and N. V. Chawla, "Few-shot graph learning for molecular property prediction," in *Proceedings of the Web Conference 2021*, 2021, pp. 2559–2567.

[36] H. Yao, C. Zhang, Y. Wei, M. Jiang, S. Wang, J. Huang, N. Chawla, and Z. Li, "Graph few-shot learning via knowledge transfer," in *AAAI*, vol. 34, no. 04, 2020, pp. 6656–6663.

[37] A. S. Maiya and T. Y. Berger-Wolf, "Sampling community structure," in *Proceedings of the 19th international conference on World wide web*, 2010, pp. 701–710.

[38] W. Lin, H. Lan, and B. Li, "Generative causal explanations for graph neural networks," in *ICML*. PMLR, 2021, pp. 6666–6679.

[39] J. Yu, T. Xu, Y. Rong, Y. Bian, J. Huang, and R. He, "Graph information bottleneck for subgraph recognition," *arXiv preprint arXiv:2010.05563*, 2020.

[40] A. K. McCallum, K. Nigam, J. Rennie, and K. Seymore, "Automating the construction of internet portals with machine learning," *Information Retrieval*, vol. 3, no. 2, pp. 127–163, 2000.

[41] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, "Simplifying graph convolutional networks," in *ICML*. PMLR, 2019, pp. 6861–6871.

[42] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[43] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *arXiv preprint arXiv:1810.00826*, 2018.

[44] G. Wu, S. Gong, and P. Li, "Striking a balance between stability and plasticity for class-incremental learning," in *CVPR*, 2021, pp. 1124–1133.

[45] H. Wang, Y. Wang, Z. Zhou, X. Ji, D. Gong, J. Zhou, Z. Li, and W. Liu, "Cosface: Large margin cosine loss for deep face recognition," in *CVPR*, 2018, pp. 5265–5274.