Topology-aware Embedding Memory for Continual Learning on Expanding Networks

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

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

Yixin Chen
chen@cse.wustl.edu
Washington University in St. Louis
St. Louis, MO, USA

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

ABSTRACT

Memory replay based techniques have shown great success for continual learning with incrementally accumulated Euclidean data. Directly applying them to continually expanding networks, however, leads to the potential memory explosion problem due to the need to buffer representative nodes and their associated topological neighborhood structures. To this end, we systematically analyze the key challenges in the memory explosion problem, and present a general framework, *i.e.*, Parameter Decoupled Graph Neural Networks (PDGNNs) with Topology-aware Embedding Memory (TEM), to tackle this issue. The proposed framework not only reduces the memory space complexity from $O(nd^L)$ to $O(n)$ [1], but also fully utilizes the topological information for memory replay. Specifically, PDGNNs decouple trainable parameters from the computation ego-subnetwork via *Topology-aware Embeddings* (TEs), which compress ego-subnetworks into compact vectors (*i.e.*, TEs) to reduce the memory consumption. Based on this framework, we discover a unique *pseudo-training effect* in continual learning on expanding networks and this effect motivates us to develop a novel *coverage maximization sampling* strategy that can enhance the performance with a tight memory budget. Thorough empirical studies demonstrate that, by tackling the memory explosion problem and incorporating topological information into memory replay, PDGNNs with TEM significantly outperform state-of-the-art techniques, especially in the challenging class-incremental setting.

CCS CONCEPTS

• **Computing methodologies** → **Supervised learning**; **Neural networks**.

KEYWORDS

Expanding Networks, Expanding Graphs, Continual Graph Learning, Continual Learning

[1] n: memory budget, d: average node degree, L: the radius of the GNN receptive field

KDD '24, August 25–29, 2024, Barcelona, Spain

ACM ISBN 979-8-4007-0490-1/24/08
https://doi.org/10.1145/3637528.3671732

ACM Reference Format:
Xikun Zhang, Dongjin Song, Yixin Chen, and Dacheng Tao. 2024. Topology-aware Embedding Memory for Continual Learning on Expanding Networks. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '24), August 25–29, 2024, Barcelona, Spain.* ACM, Barcelona, Spain, 12 pages. https://doi.org/10.1145/3637528.3671732

1 INTRODUCTION

Traditional machine learning techniques for networks typically assume the types of nodes and their associated edges to be static [2]. However, real-world networks often expand constantly with emerging new types of nodes and their associated edges. Consequently, models trained incrementally on the new node types may experience catastrophic forgetting (severe performance degradation) on the old ones as shown in Figure 1. Targeting this challenge, continual learning on expanding networks [49, 97, 104] has attracted increasingly more attention recently. It exhibits enormous value in various practical applications, especially in the case where networks are relatively large, and retraining a new model over the entire network is computationally infeasible. For instance, in a social network, a community detection model has to keep adapting its parameters based on nodes from newly emerged communities; in a citation network, a document classifier needs to continuously update its parameters to distinguish the documents of newly emerged research fields.

Memory replay [4, 51, 60, 62], which stores representative examples in a buffer to retrain the model and maintain its performance over existing tasks, exhibits great success in preventing catastrophic forgetting for various continual learning tasks, *e.g.*, computer vision and reinforcement learning [3, 44, 47, 61]. Directly applying memory replay to network data with the popular message passing neural networks (MPNNs, the general framework for most GNNs) [34, 43, 68], however, could give rise to the memory explosion problem because the necessity to consider the explicit topological information of target nodes. Specifically, due to the message passing over the topological connections in networks, retraining an L-layer GNN (Figure 2, left) with n buffered nodes would require storing $O(nd^L)$ nodes [11, 15] (the number of edges is not counted yet) in the buffer, where d is the average node degree. Take the Reddit dataset [36] as an example, its average node degree is 492, and the buffer size will easily be intractable even

[2] Network is a type of graph. These two terms may be used interchangeably

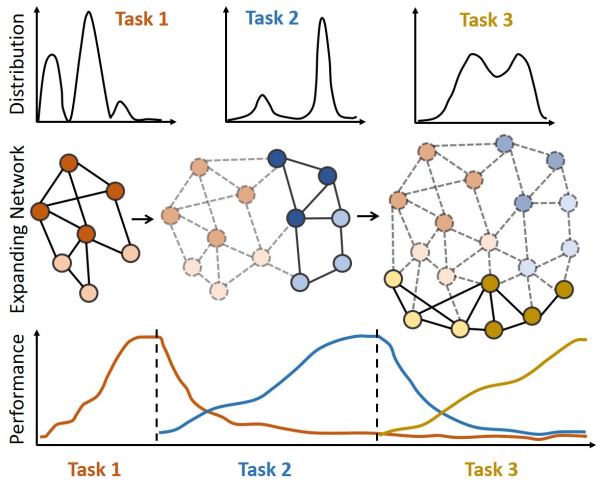

Figure 1: Learning dynamics in an expanding network. We depict new types of nodes with different colors. The new task consisting of new types of nodes may exhibit a different distribution from existing ones. Consequently, as the model adapts to these new types of nodes, it may undergo a significant performance degradation on existing tasks, a phenomenon known as catastrophic forgetting.

with a 2-layer GNN. To resolve this issue, Experience Replay based GNN (ER-GNN) [104] stores representative input nodes (*i.e.*, node attributes) in the buffer but ignores the topological information (Figure 2 a). Feature graph network (FGN) [69] implicitly encodes node proximity with the inner products between the features of the target node and its neighbors. However, the explicit topological connections are abandoned and message passing is no longer feasible on the graph. Sparsified Subgraph Memory (SSM) [96] and Subgraph Episodic Memory (SEM-curvature) [98] sparsify the computation ego-subnetworks for tractable memory consumption, which still partially sacrifices topological information, especially when the computation ego-subnetworks are large and a majority of nodes/edges is removed after sparsification (Figure 2 b).

To this end, we present a general framework of Parameter Decoupled Graph Neural Networks (PDGNNs) with Topology-aware Embedding Memory (TEM) for continual learning on expanding networks (Figure 2 c). First, we demonstrate that the necessity to store the complete computation ego-subnetworks arises from the entanglement between the trainable parameters and the individual nodes/edges (Section 3.2). Targeting this problem, we design the PDGNNs, which decouple the trainable parameters from individual nodes/edges. PDGNNs enable us to develop a novel concept, *Topology-aware Embedding* (TE), which is a vector with a fixed size but contains all necessary information for retraining PDGNNs. Such TEs are desired surrogates of computation ego-subnetworks to facilitate memory replay. After learning each task, a subset of TEs is sampled and stored in the *Topology-aware Embedding Memory* (TEM). Because the size of a TE is fixed, the space complexity of a memory buffer with size n can be dramatically reduced from $O(nd^L)$ to $O(n)$. Moreover, different from continual learning on independent data without topology (*e.g.*, images), we theoretically discover that replaying the TE of a single node incurs a *pseudo-training effect* on its neighbors, which also alleviate the forgetting problem for the other nodes in the same computation ego-subnetwork. Pseudo-training effect suggests that TEs with larger coverage ratio are more beneficial to continual learning. Based on the theoretical finding, we develop the *coverage maximization sampling* strategy, which effectively enhances the performance for a tight memory budget. In our experiments, thorough empirical studies demonstrate that PDGNNs-TEM outperform the state-of-the-art methods in both class-incremental (class-IL) [60, 95, 96] and the task-incremental (task-IL) continual learning scenarios [49, 104].

2 RELATED WORKS

2.1 Continual Learning & Continual Learning on Expanding Networks

Existing continual learning (CL) approaches can be categorized into regularization, memory replay, and parameter isolation based methods. Regularization based methods aim to prevent drastic modification to parameters that are important for previous tasks [3, 24, 33, 38, 44, 47, 48, 58, 59, 64, 65, 71, 79]. Parameter isolation methods adaptively allocate new parameters for the new tasks to protect the ones for the previous tasks [61, 77, 81, 87, 88, 92]. Memory replay based methods store and replay representative data from previous tasks when learning new tasks [4, 6, 16, 51, 60, 62, 98]. Recently, CL on expanding networks attracts increasingly more attention due to its practical importance [2, 7, 8, 18–22, 25, 30, 39, 42, 46, 50, 63, 70, 73, 76, 83, 90, 94, 95, 97, 99]. Existing methods include regularization ones like topology-aware weight preserving (TWP) [49] that preserves crucial topologies, parameter isolation methods like HPNs [97] that select different parameters for different tasks, and memory replay methods like ER-GNN [104], SSM [96], and SEM-curvature[98] that store representative nodes or sparsified computation ego-subnetworks. Our work is also memory based and its key advantage is the capability to preserve complete topological information with reduced space complexity, which shows significant superiority in class-IL setting (Section 4.4). Finally, it is worth highlighting the difference between CL on expanding networks and some relevant research areas. First, dynamic graph learning [26, 29, 37, 41, 52, 55, 72, 89, 106] focuses on the temporal dynamics with all previous data being accessible. In contrast, CL on expanding networks aims to alleviate forgetting, therefore the previous data is inaccessible. Second, few-shot graph learning [35, 66, 86, 105] targets fast adaptation to new tasks. In training, few-shot learning models can access all previous tasks (unavailable in CL). In testing, few-shot learning models need to be fine-tuned on the test classes, while the CL models are tested on existing tasks without fine-tuning.

2.2 GNNs & Reservoir Computing

Graph Neural Networks (GNNs) are deep learning models designed to generate representations for graph data, which typically interleave the neighborhood aggregation and node feature transformation to extract the topological features [10, 14, 34, 36, 43, 68, 74, 75, 80, 82, 84, 85, 93, 100, 102]. GNNs without interleaving the neighborhood aggregation and node feature transformation have been

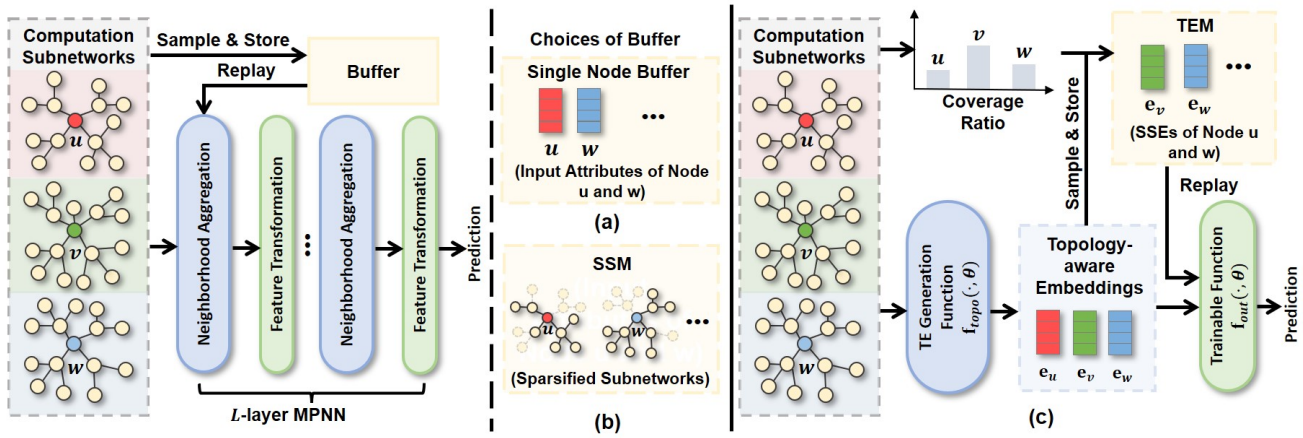

Figure 2: (a) ER-GNN [104] that stores the *input attributes* of individual nodes. (b) Sparsified Subgraph Memory (SSM) [96] that stores sparsified computation ego-subnetworks. (c) Our PDGNNs with TEM. The incoming computation ego-subnetworks are embedded as TEs and then fed into the trainable function. The stored TEs are sampled based on their coverage ratio (Section 3.7).

developed to reduce the computation complexity and increase the scalability [12, 13, 17, 23, 27, 28, 56, 91, 101]. For example, Simple Graph Convolution (SGC) [78] removes the non-linear activation from GCN [43] and only keeps one neighborhood aggregation and one node transformation layer. Approximate Personalized Propagation of Neural Predictions (APPNP) [45] first performs node transformation and then conducts multiple neighborhood aggregations in one layer. Motivated by these works, the PDGNNs framework in this paper is specially designed to decouple the neighborhood aggregation with trainable parameters, and derive the topology-aware embeddings (TEs) to reduce the memory space complexity and facilitate continual learning on expanding networks. Besides, PDGNNs are also related to reservoir computing [31, 32], which embed the input data (*e.g.* graphs) via a fixed non-linear system. The reservoir computing modules can be adopted in PDGNNs (Equation 4).

3 PARAMETER DECOUPLED GNNS WITH TOPOLOGY-AWARE EMBEDDING MEMORY

In this section, we first introduce the notations, and then explain the technical challenge of applying memory replay techniques to GNNs. Targeting the challenge, we introduce PDGNNs with Topology-aware Embedding Memory (TEM). Finally, inspired by theoretical findings of the *pseduo-training effect*, we develop the coverage maximization sampling to enhance the performance when the memory budget is tight, which has shown its effectiveness in our empirical study. All detailed proofs are provided in Appendix A.

3.1 Preliminaries

Continual learning on expanding networks is formulated as learning node representations on a sequence of subnetworks (tasks): $\mathcal{S} = \{\mathcal{G}_1, \mathcal{G}_2, ..., \mathcal{G}_T\}$. Each $\mathcal{G}_\tau$ (*i.e.*, τ-th task) contains several new categories of nodes in the overall network, and is associated with a node set $\mathbb{V}_\tau$ and an edge set $\mathbb{E}_\tau$, which is represented as the adjacency matrix $\mathbf{A}_\tau \in \mathbb{R}^{|\mathbb{V}_\tau| \times |\mathbb{V}_\tau|}$. $\mathbb{V}$ will be used to denote an arbitrary node set in the following. The degree of a node d refers to the number of edges connected to it. In practice, $\mathbf{A}_\tau$ is often 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. Each node $v \in \mathbb{V}_\tau$ has a feature vector $\mathbf{x}_v \in \mathbb{R}^b$. In classification tasks, each node v has a label $\mathbf{y}_v \in \{0, 1\}^C$, where C is the total number of classes. When generating the representation for a target node v, a L-layer GNN typically takes a computation ego-subnetwork $\mathcal{G}_{\tau,v}^{sub}$, containing the L-hop neighbors of v (*i.e.* $\mathcal{N}^L(v)$), as the input. For simplicity, $\mathcal{G}_v^{sub}$ is used in the following.

3.2 Memory Replay Meets GNNs

In traditional continual learning, a model $\mathrm{f}(\cdot; \boldsymbol{\theta})$ parameterized by $\boldsymbol{\theta}$ is sequentially trained on T tasks. Each task τ ($\tau \in \{1, ..., T\}$) corresponds to a dataset $\mathbb{D}_\tau = \{(\mathbf{x}_i, \mathbf{y}_i)_{i=1}^{n_\tau}\}$. To avoid forgetting, memory replay based methods store representative data from the old tasks in a buffer $\mathcal{B}$. When learning new tasks. A common approach to utilize $\mathcal{B}$ is through an auxiliary loss:

$$\mathcal{L} = \underbrace{\sum_{\mathbf{x}_i \in \mathbb{D}_\tau} l(\mathrm{f}(\mathbf{x}_i; \boldsymbol{\theta}), \mathbf{y}_i)}_{\mathcal{L}_\tau\text{: loss of the current task}} + \lambda \underbrace{\sum_{\mathbf{x}_j \in \mathcal{B}} l(\mathrm{f}(\mathbf{x}_j; \boldsymbol{\theta}), \mathbf{y}_j)}_{\mathcal{L}_{aux}\text{: auxiliary loss}}, \tag{1}$$

where $l(\cdot, \cdot)$ denotes the loss function, and $\lambda \geq 0$ balances the contribution of the old data. Instead of directly minimizing $\mathcal{L}_{aux}$, the buffer $\mathcal{B}$ may also be used in other ways to prevent forgetting [51, 60]. In these applications, the space complexity of a buffer containing n examples is $O(n)$.

However, to capture the topological information, GNNs obtain the representation of a node v based on a computation ego-subnetwork surrounding v. We exemplify it with the popular MPNN framework [34], which updates the hidden node representations at the $l+1$-th layer as:

$$\mathbf{m}_v^{l+1} = \sum_{w \in \mathcal{N}^1(v)} \mathrm{M}_l(\mathbf{h}_v^l, \mathbf{h}_w^l, \mathbf{x}_{v,w}^e; \boldsymbol{\theta}_l^{\mathrm{M}}), \quad \mathbf{h}_v^{l+1} = \mathrm{U}_l(\mathbf{h}_v^l, \mathbf{m}_v^{l+1}; \boldsymbol{\theta}_l^{\mathrm{U}}), \tag{2}$$

where $\mathbf{h}_v^l$, $\mathbf{h}_w^l$ are hidden representations of nodes at layer l, $\mathbf{x}_{v,w}^e$ is the edge feature, $\mathrm{M}_l(\cdot,\cdot,\cdot;\boldsymbol{\theta}_l^{\mathrm{M}})$ is the message function to integrate neighborhood information, and $\mathrm{U}_l(\cdot,\cdot;\boldsymbol{\theta}_l^{\mathrm{U}})$ updates $\mathbf{m}_v^{l+1}$ into $\mathbf{h}_v^l$ ($\mathbf{h}_v^0$ is the input features). In a L-layer MPNN, the representation of a node v can be simplified as,

$$\mathbf{h}_v^L = \mathrm{MPNN}(\mathbf{x}_v, \mathcal{G}_v^{sub}; \boldsymbol{\Theta}), \tag{3}$$

where $\mathcal{G}_v^{sub}$ contains the L-hop neighbors ($\mathcal{N}^L(v)$), $\mathrm{MPNN}(\cdot,\cdot;\boldsymbol{\Theta})$ is the composition of all $\mathrm{M}_l(\cdot,\cdot,\cdot;\boldsymbol{\theta}_l^{\mathrm{M}})$ and $\mathrm{U}_l(\cdot,\cdot;\boldsymbol{\theta}_l^{\mathrm{U}})$ at different layers. Since $\mathcal{N}^L(v)$ typically contains $O(d^L)$ nodes, replaying n nodes requires storing $O(nd^L)$ nodes (the edges are not counted yet), where d is the average degree. Therefore, the buffer size will be easily intractable in practice (*e.g.* the example of Reddit dataset in Introduction), and directly storing the computation ego-subnetworks for memory replay is infeasible for GNNs.

3.3 Parameter Decoupled GNNs with TEM

As we discussed earlier, the key challenge of applying memory replay to network data is to preserve the rich topological information of the computation ego-subnetworks with potentially unbounded sizes. Therefore, a natural resolution is to preserve the crucial topological information with a compact vector such that the memory consumption is tractable. Formally, the desired subnetwork representation can be defined as *Topology-aware Embedding* (TE).

Definition 1 (Topology-aware embedding). *Given a specific GNN parameterized with $\boldsymbol{\theta}$ and an input $\mathcal{G}_v^{sub}$, an embedding vector $\mathbf{e}_v$ is a topology-aware embedding for $\mathcal{G}_v^{sub}$ with respect to this GNN, if optimizing $\boldsymbol{\theta}$ with $\mathcal{G}_v^{sub}$ or $\mathbf{e}_v$ for this specific GNN are equivalent, i.e. $\mathbf{e}_v$ contains all necessary topological information of $\mathcal{G}_v^{sub}$ for training this GNN.*

However, TEs cannot be directly derived from the MPNNs due to their interleaved neighborhood aggregation and feature transformations. According to Section 3.2, whenever the trainable parameters get updated, recalculating the representation of a node v requires all nodes and edges in $\mathcal{G}_v^{sub}$. To resolve this issue, we formulate the Parameter Decoupled Graph Neural Networks (PDGNNs) framework, which decouples the trainable parameters from the individual nodes/edges. PDGNNs may not be the only feasible framework to derive TEs, but is the first attempt and is empirically effective. Given $\mathcal{G}_v^{sub}$, the prediction of node v with PDGNNs consists of two steps. First, the topological information of $\mathcal{G}_v^{sub}$ is encoded into an embedding $\mathbf{e}_v$ via the function $\mathrm{f}_{topo}(\cdot)$ without trainable parameters (instantiations of $\mathrm{f}_{topo}(\cdot)$ are detailed in Section 3.4).

$$\mathbf{e}_v = \mathrm{f}_{topo}(\mathcal{G}_v^{sub}). \tag{4}$$

Next, $\mathbf{e}_v$ is further passed into a trainable function $\mathrm{f}_{out}(\cdot;\boldsymbol{\theta})$ parameterized by $\boldsymbol{\theta}$ (instantiations of $\mathrm{f}_{out}(\cdot;\boldsymbol{\theta})$ are detailed in Section 3.4) to get the output prediction $\hat{\mathbf{y}}_v$,

$$\hat{\mathbf{y}}_v = \mathrm{f}_{out}(\mathbf{e}_v;\boldsymbol{\theta}). \tag{5}$$

With the formulations above, $\mathbf{e}_v$ derived in Eq. (4) clearly satisfies the requirements of TE (Definition 1). Specifically, since the trainable parameters acts on $\mathbf{e}_v$ instead of any individual node/edge, optimizing the model parameters $\boldsymbol{\theta}$ with either $\mathbf{e}_v$ or $\mathcal{G}_v^{sub}$ are equivalent. Therefore, to retrain the model, the memory buffer only needs to store TEs instead of the original computation ego-subnetworks, which reduces the space complexity from $O(nd^L)$ to $O(n)$. We name the buffer to store the TEs as *Topology-aware Embedding Memory* ($\mathcal{TEM}$). Given a new task τ, the update of $\mathcal{TEM}$ is:

$$\mathcal{TEM} = \mathcal{TEM} \bigcup \mathrm{sampler}(\{\mathbf{e}_v \mid v \in \mathbb{V}_\tau\}, n), \tag{6}$$

where $\mathrm{sampler}(\cdot,\cdot)$ is the adopted sampling strategy to populate the buffer, $\bigcup$ denotes the set union, and n is the budget. According to the experimental results (Section 4.3), as long as $\mathcal{TEM}$ is maintained, PDGNNs-TEM can perform reasonably well with different choices of $\mathrm{sampler}(\cdot,\cdot)$, including the random sampling. Nevertheless, in Section 3.7, based on the theoretical insights in Section 3.5, we propose a novel sampling strategy to better populate $\mathcal{TEM}$ when the memory budget is tight, which is empirically verified to be effective in Section 4.3. Besides, Equation (6) assumes that all data of the current task are presented concurrently. In practice, the data of a task may come in multiple batches (*e.g.*, nodes come in batches on large networks), and the buffer update have to be slightly modified by either storing sizes of the computational ego-networks and recalculating the multinomial distribution or adopting reservoir sampling. For task τ with network $\mathcal{G}_\tau$, the loss with $\mathcal{TEM}$ then becomes:

$$\mathcal{L} = \underbrace{\sum_{v\in\mathbb{V}_\tau} l(\mathrm{f}_{out}(\mathbf{e}_v;\boldsymbol{\theta}), \mathbf{y}_v)}_{\mathcal{L}_\tau\text{: loss of the current task } \tau} + \lambda \underbrace{\sum_{\mathbf{e}_w\in\mathcal{TEM}} l(\mathrm{f}_{out}(\mathbf{e}_w;\boldsymbol{\theta}), \mathbf{y}_w)}_{\mathcal{L}_{aux}\text{: auxiliary loss}}. \tag{7}$$

λ balances the contribution of the data from the current task and the memory, and is typically manually chosen in traditional continual learning works. However, on network data, we adopt a different strategy to re-scale the losses according to the class sizes to counter the bias from the severe class imbalance, which cannot be handled on networks by directly balancing the datasets.

3.4 Instantiations of PDGNNs

Although without trainable parameters, the function $\mathrm{f}_{topo}(\cdot)$ for generating TEs can be highly expressive with various formulations including linear and non-linear ones, both of which are studied in this work. First, the linear instantiations of $\mathrm{f}_{topo}(\cdot)$ can be generally formulated as,

$$\mathbf{e}_v = \mathrm{f}_{topo}(\mathcal{G}_v^{sub}) = \sum_{w\in\mathbb{V}} \mathbf{x}_w \cdot \pi(v,w;\hat{\mathbf{A}}), \tag{8}$$

where $\pi(\cdot,\cdot;\hat{\mathbf{A}})$ denotes the strategy for computation ego-subnetwork construction and determines how would the model capture the topological information. Equation (8) describes the operation on each node. In practice, Equation (8) could be implemented as matrix multiplication to generate TEs of a set of nodes $\mathbb{V}$ in parallel, *i.e.* $\mathbf{E}_\mathbb{V} = \boldsymbol{\Pi}\mathbf{X}_\mathbb{V}$, where each entry $\boldsymbol{\Pi}_{v,w} = \pi(v,w;\hat{\mathbf{A}})$. $\mathbf{E}_\mathbb{V} \in \mathbb{R}^{|\mathbb{V}|\times b}$ is the concatenation of all TEs ($\mathbf{e}_v \in \mathbb{R}^b$), and $\mathbf{X}_\mathbb{V} \in \mathbb{R}^{|\mathbb{V}|\times b}$ is the concatenation of all node feature vectors $\mathbf{x}_v \in \mathbb{R}^b$. In our experiments, we adopt three representative strategies. The first strategy (S1) [78] is a basic version of message passing and can be formulated as $\boldsymbol{\Pi} = \hat{\mathbf{A}}^L$. The second strategy (S2) [107] considers balancing the contribution of neighborhood information from different hops via a

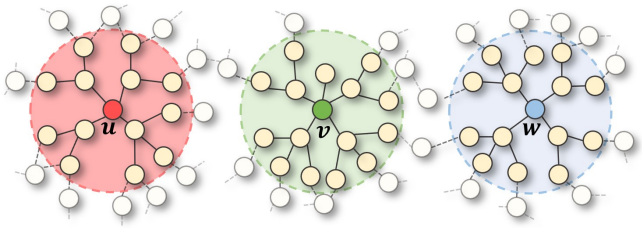

Figure 3: Illustration of the coverage ratio. Supposing the network has N nodes, $R_c(\{u\}) = \frac{13}{N}$, $R_c(\{v\}) = \frac{15}{N}$, $R_c(\{u\}) = \frac{14}{N}$, and $R_c(\{u, v, w\}) = \frac{42}{N}$

hyperparameter α, *i.e.* $\mathbf{\Pi} = \frac{1}{L}\sum_{l=1}^{L}\left((1-\alpha)\hat{\mathbf{A}}^l + \alpha\mathbf{I}\right)$. Finally, we also adopt a strategy (S3) [45] that adjusts the contribution of the neighbors based on PageRank [57], *i.e.* $\mathbf{\Pi} = \left((1-\alpha)\hat{\mathbf{A}} + \alpha\mathbf{I}\right)^L$, in which α also balances the contribution of the neighborhood information.

The linear formulation of $\mathrm{f}_{topo}(\cdot)$ (Equation (8)) yields both promising experimental results (Section 4) and instructive theoretical results (Section 3.5, and 3.7). Equation (8) is also highly efficient especially for large networks due to the absence of iterative neighborhood aggregations. But $\mathrm{f}_{topo}(\cdot)$ can also take non-linear forms with more complex mappings. For example, we can also adopt a reservoir computing module [32] to instantiate $\mathrm{f}_{topo}(\cdot)$, which is formulated as,

$$\mathrm{f}^i_{topo}(\mathcal{G}^{sub}_v) = \tanh\Big(\mathbf{W}_I \cdot \mathbf{x}^{i-1}_v + \sum_{w\in\mathbb{V}} \mathbf{W}_H \cdot \mathbf{x}^{i-1}_w\Big), i = 1, .., L, \quad (9)$$

where $\mathbf{W}_I$ and $\mathbf{W}_H$ are fixed weight matrices, and $\mathbf{e}_v = \mathrm{f}^L_{topo}(\mathcal{G}^{sub}_v)$.

Since $\mathrm{f}_{out}(\cdot;\boldsymbol{\theta})$ simply deals with individual vectors (TEs), it is instantiated as MLP in this work. The specific configurations of $\mathrm{f}_{out}(\cdot;\boldsymbol{\theta})$ is described in the experimental part (Section 4.2).

3.5 Pseudo-training Effects of TEs

In traditional continual learning on independent data without explicit topology, replaying an example $\mathbf{x}_i$ (*e.g.*, an image) only reinforces the prediction of itself. In this subsection, we introduce the pseudo-training effect, which implies that training PDGNNs with $\mathbf{e}_v$ of node v also influences the predictions of the other nodes in $\mathcal{G}^{sub}_v$, based on which we develop a novel sampling strategy to further boost the performance with a tight memory budget.

Theorem 1 (Pseudo-training). *Given a node v, its computation ego-subnetwork $\mathcal{G}^{sub}_v$, the TE $\mathbf{e}_v$, and label $\mathbf{y}_v$ (suppose v belongs to class k, i.e. $\mathbf{y}_{v,k} = 1$), then training PDGNNs with $\mathbf{e}_v$ has the following two properties:*

1. *It is equivalent to training PDGNNs with each node w in $\mathcal{G}^{sub}_v$ with $\mathcal{G}^{sub}_v$ being a pseudo computation ego-subnetwork and $\mathbf{y}_v$ being a pseudo label, where the contribution of $\mathbf{x}_w$ (via Equation 8) is re-scaled by $\frac{\pi(v,w;\hat{\mathbf{A}})}{\pi(w,w;\hat{\mathbf{A}})}$. We term this property as the pseudo-training effect on neighboring nodes, because it is equivalent to that the training is conducted on each neighboring node (in $\mathcal{G}^{sub}_v$) through the pseudo labels and the pseudo computation ego-subnetworks.*

2. *When $\mathrm{f}_{out}(\cdot;\boldsymbol{\theta})$ is linear, training PDGNNs on $\mathbf{e}_v$ is also equivalent to training $\mathrm{f}_{out}(\cdot;\boldsymbol{\theta})$ on pseudo-labeled nodes $(\mathbf{x}_w, \mathbf{y}_v)$ for each w in $\mathcal{G}^{sub}_v$, where the contribution of w in the loss is adaptively re-scaled with a weight $\frac{\mathrm{f}_{out}(\mathbf{x}_w;\boldsymbol{\theta})_k \cdot \pi(v,w;\hat{\mathbf{A}})}{\sum_{w\in\mathbb{V}^{sub}_v} \mathrm{f}_{out}\left(\mathbf{x}_w \cdot \pi(v,w;\hat{\mathbf{A}});\boldsymbol{\theta}\right)_k}$.*

The pseudo-training effect essentially arises from the neighborhood aggregation operation of GNNs, of which the rationale is to iteratively refine the node embeddings with similar neighbors. Pseudo-training effect implies that replaying the TE of one node can also strengthen the prediction for its neighbors within the same computation ego-subnetwork and alleviate the forgetting problem on them. The above analysis suggests that TEs with larger computation ego-subnetworks covering more nodes may be more effective, motivating our coverage maximization sampling strategy in the next subsection, which is also empirically justified in Section 4.3.

Algorithm 1 Coverage maximization sampling

Input: $\mathcal{G}_\tau$, $\mathbb{V}_\tau$, $\hat{\mathbf{A}}_\tau$, $\pi(\cdot,\cdot;\cdot)$, sample size n.
Output: Selected nodes $\mathcal{S}$
Initialize $\mathcal{S} = \{\}$.
for each $v \in \mathbb{V}_\tau$ **do**
  $R_c(\{v\}) = \frac{|\{w | w\in\mathcal{G}^{sub}_{\tau,v}\}|}{|\mathbb{V}_\tau|}$
end for
for each $v \in \mathbb{V}_\tau$ **do**
  $p_v = \frac{R_c(\{v\})}{\sum_{w\in\mathbb{V}_\tau} R_c(\{w\})}$
end for
while $n > 0$ **do**
  Sample one node v from $\mathbb{V}_\tau$ according to $\{p_w \mid w \in \mathbb{V}_\tau\}$.
  $\mathcal{S} = \mathcal{S} \cup \{v\}$
  $\mathbb{V}_\tau = \mathbb{V}_\tau \backslash \{v\}$ #Sampling without replacement
  $n \leftarrow n - 1$
end while

3.6 Pseudo-training Effect and Network Homophily

In this subsection, we provide a brief discussion on pseudo-training and network homophily. Given a network $\mathcal{G}$, the homophily ratio is defined as the ratio of the number of edges connecting nodes with a same label and the total number of edges, *i.e.*

$$h(\mathcal{G}) = \frac{1}{|\mathcal{E}|}\sum_{(j,k)\in\mathcal{E}} \mathbf{1}(\mathbf{y}_j = \mathbf{y}_k), \quad (10)$$

where $\mathcal{E}$ is the edge set, $\mathbf{y}_j$ is the label of node j, and $\mathbf{1}(\cdot)$ is the indicator function [53]. For any network, the homophily ratio is between 0 and 1. For each computation ego-subnetwork, when the homophily ratio is high, the neighboring nodes tend to share labels with the center node, and the pseudo training would be beneficial for the performance. Many real-world networks, *i.e.* the social network and citation networks, tend to have high homophily ratios, and pseudo training will bring much benefit,.

In our work, the homophily ratio of the 4 network datasets are: CoraFull-CL (0.567), Arxiv-CL (0.655), OGB-Products (0.807),

Table 1: The detailed statistics of datasets and task splittings

Dataset	CoraFull [54]	Arxiv [40]	Reddit [36]	Products [40]
# 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	30	20	20	23

Reddit-CL (0.755). These datasets cover the ones with high homophily (OGB-Products and Reddit), as well as the ones with lower homophily.

When learning on more heterophilous networks (homophily ratio close to 0) $f_{topo}(\cdot)$ is required to be constructed specially constructed. Heterophilous network learning is largely different from homophilous network learning, and requires different GNN designs [1, 103, 108]. Therefore, $f_{topo}(\cdot)$ should also be instantiated to be suitable for heterophilous networks. The key difference of heterophilous network learning is that the nodes belonging to the same classes are not likely to be connected, and GNNs should be designed to separately process the proximal neighbors with similar information and distal neighbors with irrelevant information, or only aggregate information from the proximal neighbors [1, 103, 108]. For example, MixHop [1] encodes neighbors from different hops separately. A given computation ego-subnetwork will be divided into different hops. For each hop, the model generates a separate embedding. Finally, the embeddings of different hops are concatenated as the final TE. H2GCN [108] only aggregates higher-order neighbors that are proximal to the center node.

In other words, via constructing $f_{topo}(\cdot)$ to be suitable for heterophilous networks, the neighborhood aggregation is still conducted on the proximal nodes, and so is the pseudo-training. In this way, the pseudo-training will still benefit the performance.

3.7 Coverage Maximization Sampling

Following the above subsection, TEs with larger computation ego-subnetworks are preferred to be stored. To quantify the size of the computation ego-subnetworks, we formally define the coverage ratio of the selected TEs as the nodes covered by their computation ego-subnetworks versus the total nodes in the network (Figure 3). Since a TE uniquely corresponds to a node, we may use 'node' and 'TE' interchangeably.

Definition 2. *Given a network $\mathcal{G}$, node set $\mathbb{V}$, and function $\pi(\cdot,\cdot;\hat{\mathbf{A}})$, the coverage ratio of a set of nodes $\mathbb{V}_s$ is:*

$$R_c(\mathbb{V}_s) = \frac{|\cup_{v\in\mathbb{V}_s}\{w|w\in\mathcal{G}_v^{sub}\}|}{|\mathbb{V}|}, \tag{11}$$

i.e., the ratio of nodes of the entire (training) network covered by the computation ego-subnetworks of the selected nodes (TEs).

To maximize $R_c(\mathcal{TEM})$, a naive approach is to first select the TE with the largest coverage ratio, and then iteratively incorporate TE that increases $R_c(\mathcal{TEM})$ the most. However, this requires computing $R_c(\mathcal{TEM})$ for all candidate TEs at each iteration, which is time consuming especially on large networks. Besides, certain randomness is also desired for the diversity of $\mathcal{TEM}$. Therefore, we propose to sample TEs based on their coverage ratio. Specifically, in task τ, the probability of sampling node $v\in\mathbb{V}_\tau$ is $p_v = \frac{R_c(\{v\})}{\sum_{w\in\mathbb{V}_\tau} R_c(\{w\})}$. Then the nodes in $\mathbb{V}_\tau$ are sampled according to $\{p_v \mid v\in\mathbb{V}_\tau\}$ without replacement, as shown in Algorithm 1. In experiments, we demonstrate the correlation between the coverage ratio and the performance, which verifies the benefits revealed in Section 3.5

4 EXPERIMENTS

In this section, we aim to answer the following research questions: RQ1: Whether PDGNNs-TEM works well with a reasonable buffer size? RQ2: Does coverage maximization sampling ensure a higher coverage ratio and better performance when the memory budget is tight? RQ3: Whether our theoretical results can be reflected in experiments? RQ4: Whether PDGNNs-TEM can outperform the state-of-the-art methods in both class-IL and task-IL scenarios? RQ5: How to interpret the learned node embedding under continual learning setting. Due to the space limitations, only the most prominent results are presented in the main content. For simplicity, PDGNNs-TEM will be denoted as PDGNNs in this section. All codes are available at github.com/imZHANGxikun/PDGNNs.

4.1 Datasets

Following the public benchmark CGLB [95], we adopted four datasets, CoraFull [54], OGB-Arxiv [40], Reddit [36], and OGB-Products [40], with up to millions of nodes and 70 classes. Dataset statistics and task splittings are summarized in Table 1.

4.2 Experimental Setup and Model Evaluation

Continual learning setting and model evaluation. During training, a model is trained on a task sequence. During testing, the model is tested on all learned tasks. Class-IL requires a model to classify a given node by picking a class from all learned classes (more challenging), while task-IL only requires the model to distinguish the classes within each task. For model evaluation, the most thorough metric is the accuracy matrix $\mathrm{M}^{acc}\in\mathbb{R}^{T\times T}$, where $\mathrm{M}^{acc}_{i,j}$ denotes the accuracy on task j after learning task i. The learning dynamics can be reflected with average accuracy (AA) over all learnt tasks after learning each task, *i.e.*, $\left\{\frac{\sum_{j=1}^{i}\mathrm{M}^{acc}_{i,j}}{i}|i=1,...,T\right\}$, which can be visualized as a curve. Similarly, the average forgetting (AF) after learning each task reflects the learning dynamics from the perspective of forgetting, $\left\{\frac{\sum_{j=1}^{i-1}\mathrm{M}^{acc}_{i,j}-\mathrm{M}^{acc}_{j,j}}{i-1}|i=2,...,T\right\}$. To use a single numeric value for evaluation, the AA and AF after learning all T tasks will be used. These metrics are widely adopted in continual learning works [9, 49, 51, 97, 104], although the names are different in different works. We repeat all experiments 5 times on one Nvidia Titan Xp GPU. All results are reported with average performance and standard deviations.

Baselines and model settings. Our baselines for continual learning on expanding networks include Experience Replay based GNN (ER-GNN) [104], Topology-aware Weight Preserving (TWP) [49], Sparsified Subgraph Memory (SSM) [96], and Subgraph Episodic Memory (SEM) [98]. Milestone works for Euclidean data but also

Table 2: Performance & coverage ratios of different sampling strategies and buffer sizes on OGB-Arxiv (↑ higher means better).

Ratio of dataset /%		0.02	0.1	1.0	5.0	40.0
AA/%	Uniform samp.	12.0±1.1	24.1±1.7	42.2±0.3	50.4±0.4	53.3±0.4
	Mean of feat.	12.6±0.1	25.3±0.3	42.8±0.3	50.4±0.7	53.3±0.2
	Cov. Max.	**14.9±0.8**	**26.8±1.8**	**43.7±0.5**	**50.5±0.4**	**53.4±0.1**
Cov. ratio/%	Uniform samp.	0.1±0.1	0.3±0.0	3.5±0.9	15.9±1.1	84.8±1.5
	Mean of feat.	0.2±0.4	0.6±0.3	7.1±0.6	29.6±1.7	91.1±0.1
	Cov. Max.	**0.5±1.1**	**2.9±1.8**	**22.5±1.6**	**46.3±0.6**	**92.8±0.0**

Table 3: Performance comparisons under class-IL on different datasets (↑ higher means better).

C.L.T.	CoraFull		OGB-Arxiv		Reddit		OGB-Products	
	AA/% ↑	AF/% ↑	AA/% ↑	AF /% ↑	AA/% ↑	AF /% ↑	AA/% ↑	AF /% ↑
Fine-tune	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 (2017)	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 (2018)	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 (2017)	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 (2021)	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 (2017)	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 (2021)	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 (2022b)	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
SEM-curvature (2023b)	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
Joint	80.6±0.3	-	46.4±1.4	-	99.3±0.2	-	71.5±0.7	-
PDGNNs	**81.9±0.1**	-3.9±0.1	**53.2±0.2**	-14.7±0.2	94.7±0.4	-3.0±0.4	**73.9±0.1**	-10.9±0.2

applicable to GNNs include Elastic Weight Consolidation (EWC) [44], Learning without Forgetting (LwF) [47], Gradient Episodic Memory (GEM) [51], and Memory Aware Synapses (MAS) [3]), are also adopted. HPNs [97] is designed to work under a stricter task-IL setting, and cannot be properly incorporated for comparison. The results of the baselines are adopted from the original works [95, 96, 98]. Besides, joint training (without forgetting problem) and fine-tune (without continual learning technique) are adopted as the upper and lower bound on the performance. We instantiate $f_{out}(\cdot;\theta)$ as a multi-layer perceptron (MLP). All methods including $f_{out}(\cdot;\theta)$ of PDGNNs are set as 2-layer with 256 hidden dimensions, and L in Section 3.3 is set as 2 for consistency. As detailed in Section 4.3, $f_{topo}(\cdot)$ is chosen as strategy S1 (Section 3.4).

4.3 Studies on the Buffer Size & Performance vs. Coverage Ratio (RQ1, 2, and 3)

In Table 2, based on PDGNNs, we compare the proposed *coverage maximization sampling* with uniform sampling and mean of feature (MoF) sampling in terms of coverage ratios and performance when the buffer size (ratio of the dataset) varies from 0.0002 to 0.4 on the OGB-Arxiv dataset. Our proposed *coverage maximization sampling* achieves a superior coverage ratio, which indeed enhances the performance when the memory budget is tight. In real-world applications, a tight memory budget is a very common situation, making the coverage maximization sampling a favorable choice. We also notice that the average accuracy for *coverage maximization sampling* is positively related to the coverage ratio in general, which is consistent with the Theorem 1.

Table 2 also demonstrates the high memory efficiency of TEM. No matter which sampling strategy is used, the performance can reach ≈50% average accuracy (AA) with only 5% data buffered. In Section 4.5, we provide the comparison of the space consumption of different memory based strategies to demonstrate the efficiency of PDGNNs-TEM.

Table 4: Additional space consumption of different memory-replay techniques

C.L.T.	CoraFull	OGB-Arxiv	Reddit	OGB-Products
Full Subnetwork	7,264M	35M	2,184,957M	5,341M
GEM [51]	7,840M	86M	329M	82M
ER-GNN [104]	61M	2M	12M	3M
SSM [96]	732M	41M	193M	37M
SEM [98]	732M	41M	193M	37M
PDGNNs-TEM	37M	2M	9M	2M

4.4 Class-IL and Task-IL Scenarios (RQ4)

Class-IL Scenario. As shown in Table 3, under the class-IL scenario, PDGNNs significantly outperform the baselines and are even comparable to joint training on all 4 public datasets. The learning dynamics are shown in Figure 4. Since the curve of PDGNNs is very close to that of joint training, we conclude that the forgetting

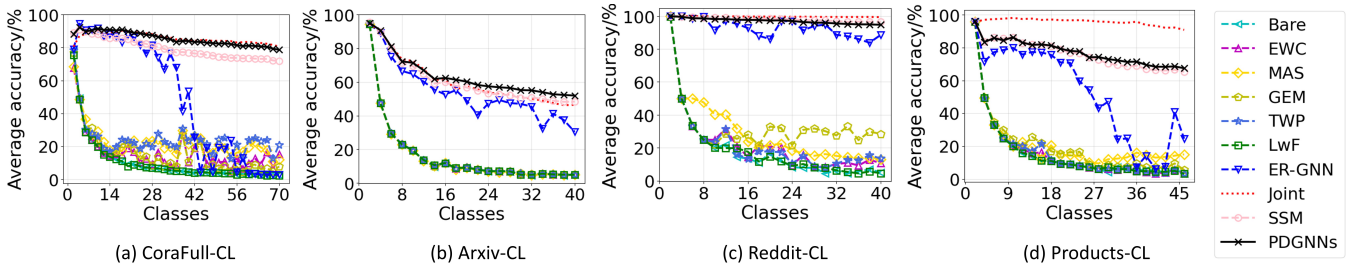

Figure 4: Dynamics of average accuracy in the class-IL scenario.(a) CoraFull, 2 classes per task, 35 tasks. (b) OGB-Arxiv, 2 classes per task, 20 tasks. (c) Reddit, 2 classes per task, 20 tasks. (d) OGB-Products, 2 classes per task, 23 tasks.

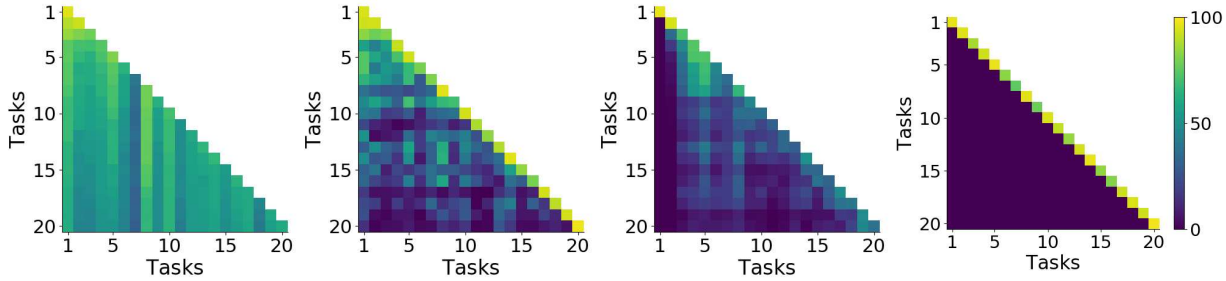

Figure 5: From left to right: accuracy matrix of PDGNNs, ER-GNN, LwF, and Fine-tune on OGB-Arxiv dataset.

Table 5: Performance comparisons under task-IL on different datasets (↑ higher means better).

C.L.T.	CoraFull		OGB-Arxiv		Reddit		OGB-Products	
	AA/% ↑	AF/% ↑	AA/% ↑	AF /% ↑	AA/% ↑	AF /% ↑	AA/% ↑	AF /% ↑
Fine-tune	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 (2017)	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 (2018)	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 (2017)	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 (2021)	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 (2017)	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 (2021)	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 (2022b)	95.8±0.3	0.6±0.2	88.4±0.3	-1.1±0.1	99.3±0.0	-0.2±0.0	93.2±0.7	-1.9±0.0
SEM-curvature (2023b)	**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
Joint	95.2±0.2	-	90.3±0.2	-	99.4±0.1	-	91.8±0.2	-
PDGNNs	94.6±0.1	0.6±1.0	89.8±0.4	-0.0±0.5	98.9±0.0	-0.5±0.0	**93.5±0.5**	-2.1±0.1

problem is nearly eliminated by PDGNNs. In Table 3 and Figure 4, PDGNNs sometimes outperform joint training. The reasons are two-fold. First, PDGNNs learn the tasks sequentially while joint training optimizes the model for all tasks simultaneously, resulting in different optimization difficulties [5]. Second, when learning new tasks, joint training accesses all previous data that may be noisy, while replaying the representative TEs may help filter out noise. To thoroughly understand different methods, we visualize the accuracy matrices of 4 representative methods, including PDGNNs (memory replay with topological information), ER-GNN (memory replay without topological information), LwF (relatively satisfying performance without memory buffer), and Fine-tune (without continual learning technique), in Figure 5. Compared to the baselines, PDGNNs maintain stable performance on each task even though new tasks are continuously learned.

Task-IL Scenario. The comparison results under the task-IL scenario are shown in Table 5. We can observe that PDGNNs still outperform most baselines on all different datasets and is comparable to SEM [98] and SSM [96], even though task-IL is less challenging than the class-IL as we discussed in Section 4.2.

4.5 Memory Consumption Comparison (RQ1)

Memory-replay based methods outperform other methods, but also consume additional memory space. In this subsection, we compare the space consumption of different memory designs to demonstrate

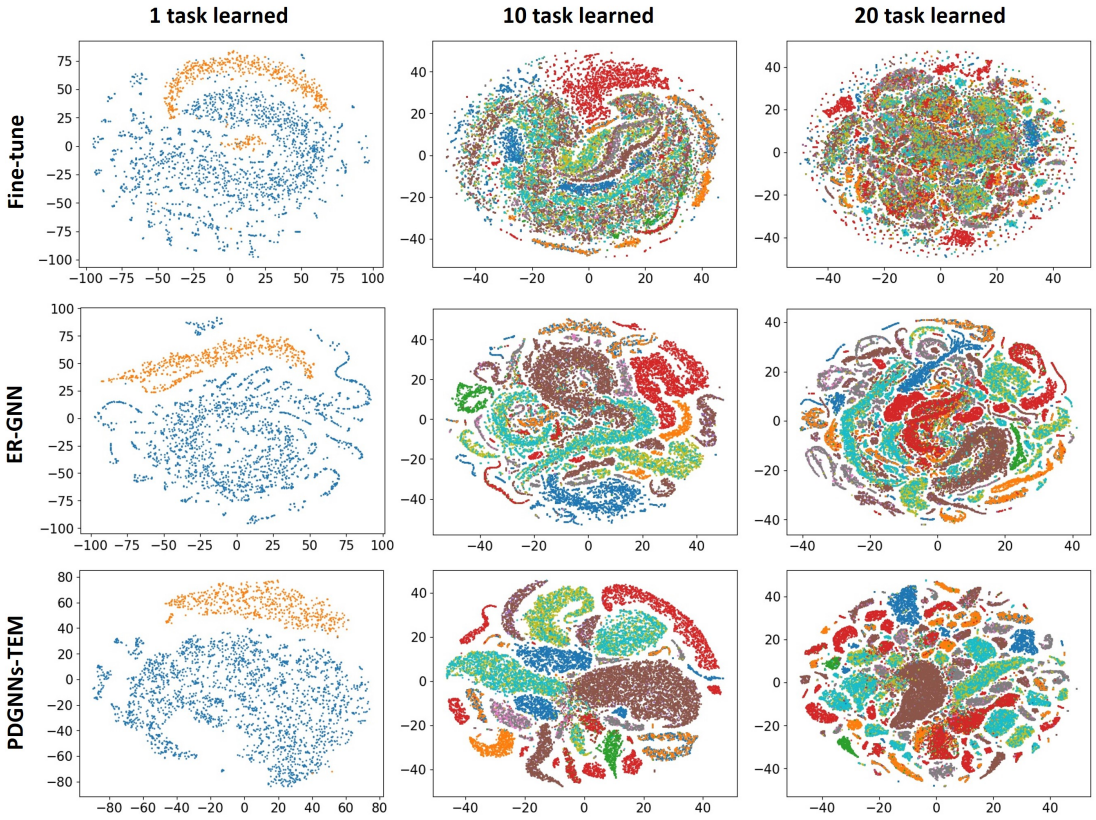

Figure 6: Visualization of the node embeddings of different classes of Reddit, after learning 1, 10, and 20 tasks. From the top to the bottom, we show the results of Fine-tune, ER-GNN, and PDGNNs-TEM. Each color corresponds to a class.

the memory efficiency of PDGNNs-TEM. The final memory consumption (measured by the number of float32 values) after learning each entire dataset is shown in Table 4. As a reference, the memory consumption of storing full computation ego-subnetwork is also calculated. According to Table 4, storing full subnetworks costs intractable memory usage on dense networks like Reddit, and the strategy to buffer gradients also incurs high memory cost (GEM). SSM could significantly reduce memory consumption with the sparsification strategy. Both PDGNNs-TEM and ER-GNN are highly efficient in terms of memory space usage. While PDGNNs-TEM exhibits superior performance compared to ER-GNN.

4.6 Interpretation of Node Embeddings (RQ5)

To interpret the learning process of PDGNNs-TEM, we visualize the node embeddings of different classes with t-SNE [67] while learning on a task sequence of 20 tasks over the Reddit dataset. In Figure 6, besides PDGNNs-TEM that replay data with topological information, we also include two representative baselines for comparison, *i.e.*, ER-GNN to show how the lack of topological information may affect the node embeddings, and Fine-tune to show the results without any continual learning technique. As shown in Figure 6, PDGNNs-TEM can well separate the nodes from different classes even when node types of nodes are continuously been involved (in new tasks). In contrast, for ER-GNN and Fine-tune, the boundaries of different classes are less clear, especially when more tasks are continuously learned.

5 CONCLUSION

In this work, we propose a general framework of Parameter Decoupled Graph Neural Networks (PDGNNs) with Topology-aware Embedding Memory (TEM) for continual learning on expanding networks. Based on the Topology-aware Embeddings (TEs), we reduce the space complexity of the memory buffer from $O(nd^L)$ to $O(n)$, which enables PDGNNs to fully utilize the explicit topological information sampled from the previous tasks for retraining. We also discover and theoretically analyze the pseudo-training effect of TEs. The theoretical findings inspire us to develop the *coverage maximization sampling* strategy, which has been demonstrated to be highly efficient when the memory budget is tight. Finally, thorough empirical studies, including comparison with the state-of-the-art methods in both class-IL and task-IL continual learning scenarios, demonstrate the effectiveness of PDGNNs with TEM.

6 ACKNOWLEDGEMENT

Dongjin Song was supported by the National Science Foundation (NSF) Grant No. 2338878. Yixin Chen is supported by NSF grant CBE-2225809 and DOE grant DE-SC0024702.

REFERENCES

[1] Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. 2019. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *ICML*. PMLR, 21–29.

[2] Kian Ahrabian, Yishi Xu, Yingxue Zhang, Jiapeng Wu, Yuening Wang, and Mark Coates. 2021. Structure aware experience replay for incremental learning in graph-based recommender systems. In *Proceedings of the 30th ACM CIKM*. 2832–2836.

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

[4] Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. 2019. Gradient based sample selection for online continual learning. In *NeurIPS*. 11816–11825.

[5] S Divakar Bhat, Biplab Banerjee, Subhasis Chaudhuri, and Avik Bhattacharya. 2021. CILEA-NET: Curriculum-based incremental learning framework for remote sensing image classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 14 (2021), 5879–5890.

[6] Lucas Caccia, Eugene Belilovsky, Massimo Caccia, and Joelle Pineau. 2020. Online learned continual compression with adaptive quantization modules. In *ICML*. PMLR, 1240–1250.

[7] Jie Cai, Xin Wang, Chaoyu Guan, Yateng Tang, Jin Xu, Bin Zhong, and Wenwu Zhu. 2022. Multimodal continual graph learning with neural architecture search. In *Proceedings of the ACM Web Conference 2022*. 1292–1300.

[8] Antonio Carta, Andrea Cossu, Federico Errica, and Davide Bacciu. 2021. Catastrophic Forgetting in Deep Graph Networks: an Introductory Benchmark for Graph Classification. *arXiv preprint arXiv:2103.11750* (2021).

[9] Arslan Chaudhry, Puneet K Dokania, Thalaiyasingam Ajanthan, and Philip HS Torr. 2018. Riemannian walk for incremental learning: Understanding forgetting and intransigence. In *Proceedings of the ECCV*. 532–547.

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

[11] Jianfei Chen, Jun Zhu, and Le Song. 2017. Stochastic training of graph convolutional networks with variance reduction. *arXiv preprint arXiv:1710.10568* (2017).

[12] Lei Chen, Zhengdao Chen, and Joan Bruna. 2020. On graph neural networks versus graph-augmented mlps. *arXiv preprint arXiv:2010.15116* (2020).

[13] Ting Chen, Song Bian, and Yizhou Sun. 2019. Are powerful graph neural nets necessary? a dissection on graph classification. *arXiv preprint arXiv:1905.04579* (2019).

[14] Wei Chen, Yajun Wang, and Siyu Yang. 2009. Efficient influence maximization in social networks. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. 199–208.

[15] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 257–266.

[16] Aristotelis Chrysakis and Marie-Francine Moens. 2020. Online continual learning from imbalanced data. In *ICML*. PMLR, 1952–1961.

[17] Weilin Cong, Rana Forsati, Mahmut Kandemir, and Mehrdad Mahdavi. 2020. Minimal variance sampling with provable guarantees for fast training of graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1393–1403.

[18] Yuanning Cui, Yuxin Wang, Zequn Sun, Wenqiang Liu, Yiqiao Jiang, Kexin Han, and Wei Hu. 2023. Lifelong embedding learning and transfer for growing knowledge graphs. In *Proceedings of AAAI*, Vol. 37. 4217–4224.

[19] Angel Daruna, Mehul Gupta, Mohan Sridharan, and Sonia Chernova. 2021. Continual learning of knowledge graph embeddings. *IEEE Robotics and Automation Letters* 6, 2 (2021), 1128–1135.

[20] Bishwadeep Das and Elvin Isufi. 2022. Graph filtering over expanding graphs. In *2022 IEEE Data Science and Learning Workshop (DSLW)*. IEEE, 1–8.

[21] Bishwadeep Das and Elvin Isufi. 2022. Learning expanding graphs for signal interpolation. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 5917–5921.

[22] Bishwadeep Das and Elvin Isufi. 2022. Online filtering over expanding graphs. In *2022 56th Asilomar Conference on Signals, Systems, and Computers*. IEEE, 43–47.

[23] Hande Dong, Jiawei Chen, Fuli Feng, Xiangnan He, Shuxian Bi, Zhaolin Ding, and Peng Cui. 2021. On the equivalence of decoupled graph convolution network and label propagation. In *Proceedings of the Web Conference 2021*. 3651–3662.

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

[25] Falih Gozi Febrinanto, Feng Xia, Kristen Moore, Chandra Thapa, and Charu Aggarwal. 2023. Graph lifelong learning: A survey. *IEEE Computational Intelligence Magazine* 18, 1 (2023), 32–51.

[26] Yutong Feng, Jianwen Jiang, and Yue Gao. 2020. Incremental Learning on Growing Graphs. (2020).

[27] Matthias Fey, Jan E Lenssen, Frank Weichert, and Jure Leskovec. 2021. Gnnautoscale: Scalable and expressive graph neural networks via historical embeddings. In *ICML*. PMLR, 3294–3304.

[28] Fabrizio Frasca, Emanuele Rossi, Davide Eynard, Ben Chamberlain, Michael Bronstein, and Federico Monti. 2020. Sign: Scalable inception graph neural networks. *arXiv preprint arXiv:2004.11198* (2020).

[29] Lukas Galke, Benedikt Franke, Tobias Zielke, and Ansgar Scherp. 2021. Lifelong Learning of Graph Neural Networks for Open-World Node Classification. In *2021 IJCNN*. 1–8. https://doi.org/10.1109/IJCNN52387.2021.9533412

[30] Lukas Galke, Iacopo Vagliano, Benedikt Franke, Tobias Zielke, Marcel Hoffmann, and Ansgar Scherp. 2023. Lifelong learning on evolving graphs under the constraints of imbalanced classes and new classes. *Neural Networks* 164 (2023), 156–176. https://doi.org/10.1016/j.neunet.2023.04.022

[31] Claudio Gallicchio and Alessio Micheli. 2010. Graph echo state networks. In *The 2010 IJCNN*. IEEE, 1–8.

[32] Claudio Gallicchio and Alessio Micheli. 2020. Fast and deep graph neural networks. In *Proceedings of AAAI*, Vol. 34. 3898–3905.

[33] Spyros Gidaris and Nikos Komodakis. 2018. Dynamic few-shot visual learning without forgetting. In *Proceedings of the IEEE CVPR*. 4367–4375.

[34] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. In *ICML*. PMLR, 1263–1272.

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

[36] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NeurIPS*. 1024–1034.

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

[38] Tyler L Hayes and Christopher Kanan. 2020. Lifelong machine learning with deep streaming linear discriminant analysis. In *Proceedings of the IEEE/CVF CVPR workshops*. 220–221.

[39] Marcel Hoffmann, Lukas Galke, and Ansgar Scherp. 2023. Open-World Lifelong Graph Learning. In *2023 IJCNN*. 1–9. https://doi.org/10.1109/IJCNN54540.2023.10191071

[40] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open graph benchmark: Datasets for machine learning on graphs. *NeurIPS* 33 (2020), 22118–22133.

[41] Yushan Jiang, Zijie Pan, Xikun Zhang, Sahil Garg, Anderson Schneider, Yuriy Nevmyvaka, and Dongjin Song. 2024. Empowering Time Series Analysis with Large Language Models: A Survey. *arXiv preprint arXiv:2402.03182* (2024).

[42] Seoyoon Kim, Seongjun Yun, and Jaewoo Kang. 2022. Dygrain: An incremental learning framework for dynamic graphs. In *31st International Joint Conference on Artificial Intelligence, IJCAI*. 3157–3163.

[43] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*. https://openreview.net/forum?id=SJU4ayYgl

[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. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences* 114, 13 (2017), 3521–3526.

[45] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997* (2018).

[46] Guozheng Li, Peng Wang, Qiqing Luo, Yanhe Liu, and Wenjun Ke. 2023. Online Noisy Continual Relation Learning. *Proceedings of AAAI* 37, 11 (Jun. 2023), 13059–13066. https://doi.org/10.1609/aaai.v37i11.26534

[47] Zhizhong Li and Derek Hoiem. 2017. Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40, 12 (2017), 2935–2947.

[48] Hongxiang Lin, Ruiqi Jia, and Xiaoqing Lyu. 2023. Gated Attention with Asymmetric Regularization for Transformer-based Continual Graph Learning. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2021–2025.

[49] Huihui Liu, Yiding Yang, and Xinchao Wang. 2021. Overcoming catastrophic forgetting in graph neural networks. In *Proceedings of AAAI*, Vol. 35. 8653–8661.

[50] Yilun Liu, Ruihong Qiu, and Zi Huang. 2023. CaT: Balanced Continual Graph Learning with Graph Condensation. *arXiv preprint arXiv:2309.09455* (2023).

[51] David Lopez-Paz and Marc'Aurelio Ranzato. 2017. Gradient episodic memory for continual learning. In *NeurIPS*. 6467–6476.

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

[53] Yao Ma, Xiaorui Liu, Neil Shah, and Jiliang Tang. 2021. Is homophily a necessity for graph neural networks? *arXiv preprint arXiv:2106.06134* (2021).

[54] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. 2000. Automating the construction of internet portals with machine learning.

Information Retrieval 3, 2 (2000), 127–163.
[55] Giang Hoang Nguyen, John Boaz Lee, Ryan A Rossi, Nesreen K Ahmed, Eunyee Koh, and Sungchul Kim. 2018. Continuous-time dynamic network embeddings. In *Companion Proceedings of the The Web Conference 2018*. 969–976.
[56] Hoang Nt and Takanori Maehara. 2019. Revisiting graph neural networks: All we have is low-pass filters. *arXiv preprint arXiv:1905.09550* (2019).
[57] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank citation ranking: Bringing order to the web.* Technical Report. Stanford InfoLab.
[58] Daiqing Qi, Handong Zhao, Yun Fu, and Sheng Li. [n. d.]. Data-Free Continual Graph Learning. ([n. d.]).
[59] Appan Rakaraddi, Lam Siew Kei, Mahardhika Pratama, and Marcus De Carvalho. 2022. Reinforced Continual Learning for Graphs. In *Proceedings of the 31st ACM CIKM*. 1666–1674.
[60] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. 2017. iCaRL: Incremental classifier and representation learning. In *Proceedings of the IEEE CVPR*. 2001–2010.
[61] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. 2016. Progressive neural networks. *arXiv preprint arXiv:1606.04671* (2016).
[62] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. 2017. Continual learning with deep generative replay. In *NeurIPS*. 2990–2999.
[63] Junwei Su and Chuan Wu. 2023. Towards Robust Inductive Graph Incremental Learning via Experience Replay. *arXiv preprint arXiv:2302.03534* (2023).
[64] Li Sun, Junda Ye, Hao Peng, Feiyang Wang, and S Yu Philip. 2023. Self-supervised continual graph learning in adaptive riemannian spaces. In *Proceedings of AAAI*, Vol. 37. 4633–4642.
[65] Li Sun, Junda Ye, Hao Peng, Feiyang Wang, and Philip S Yu. 2022. Self-Supervised Continual Graph Learning in Adaptive Riemannian Spaces. *arXiv preprint arXiv:2211.17068* (2022).
[66] Zhen Tan, Kaize Ding, Ruocheng Guo, and Huan Liu. 2022. Graph few-shot class-incremental learning. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*. 987–996.
[67] Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of Machine Learning Research* 9, 11 (2008).
[68] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).
[69] Chen Wang, Yuheng Qiu, and Sebastian Scherer. 2020. Bridging graph network to lifelong learning with feature interaction. (2020).
[70] Chen Wang, Yuheng Qiu, and Sebastian Scherer. 2020. Lifelong graph learning. *arXiv preprint arXiv:2009.00647* (2020).
[71] Hao Wang, Yitong Wang, Zheng Zhou, Xing Ji, Dihong Gong, Jingchao Zhou, Zhifeng Li, and Wei Liu. 2018. Cosface: Large margin cosine loss for deep face recognition. In *Proceedings of the IEEE CVPR*. 5265–5274.
[72] Junshan Wang, Guojie Song, Yi Wu, and Liang Wang. 2020. Streaming graph neural networks via continual learning. In *Proceedings of the 29th ACM CIKM*. 1515–1524.
[73] Junshan Wang, Wenhao Zhu, Guojie Song, and Liang Wang. 2022. Streaming graph neural networks with generative replay. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 1878–1888.
[74] Kuansan Wang, Zhihong Shen, Chiyuan Huang, Chieh-Han Wu, Yuxiao Dong, and Anshul Kanakia. 2020. Microsoft academic graph: When experts are not enough. *Quantitative Science Studies* 1, 1 (2020), 396–413.
[75] Xiaoyang Wang, Ying Zhang, Wenjie Zhang, Xuemin Lin, and Chen Chen. 2016. Bring order into the samples: A novel scalable method for influence maximization. *IEEE TKDE* 29, 2 (2016), 243–256.
[76] Di Wei, Yu Gu, Yumeng Song, Zhen Song, Fangfang Li, and Ge Yu. 2022. IncreGNN: Incremental Graph Neural Network Learning by Considering Node and Parameter Importance. In *International Conference on Database Systems for Advanced Applications*. Springer, 739–746.
[77] Mitchell Wortsman, Vivek Ramanujan, Rosanne Liu, Aniruddha Kembhavi, Mohammad Rastegari, Jason Yosinski, and Ali Farhadi. 2020. Supermasks in superposition. *arXiv preprint arXiv:2006.14769* (2020).
[78] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying graph convolutional networks. In *ICML*. PMLR, 6861–6871.
[79] Guile Wu, Shaogang Gong, and Pan Li. 2021. Striking a balance between stability and plasticity for class-incremental learning. In *Proceedings of ICCV*. 1124–1133.
[80] Xinyu Wu, Ye Yuan, Xikun Zhang, Can Wang, Tiantian Xu, and Dacheng Tao. 2021. Gait phase classification for a lower limb exoskeleton system based on a graph convolutional network model. *IEEE Transactions on Industrial Electronics* 69, 5 (2021), 4999–5008.
[81] Yue Wu, Yinpeng Chen, Lijuan Wang, Yuancheng Ye, Zicheng Liu, Yandong Guo, and Yun Fu. 2019. Large scale incremental learning. In *Proceedings of the IEEE/CVF CVPR*. 374–382.
[82] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* (2018).
[83] Yishi Xu, Yingxue Zhang, Wei Guo, Huifeng Guo, Ruiming Tang, and Mark Coates. 2020. Graphsail: Graph structure aware incremental learning for recommender systems. In *Proceedings of the 29th ACM CIKM*. 2861–2868.
[84] Yujun Yan, Milad Hashemi, Kevin Swersky, Yaoqing Yang, and Danai Koutra. 2022. Two sides of the same coin: Heterophily and oversmoothing in graph convolutional neural networks. In *2022 IEEE ICDM*. IEEE, 1287–1292.
[85] Xiaocheng Yang, Mingyu Yan, Shirui Pan, Xiaochun Ye, and Dongrui Fan. 2023. Simple and efficient heterogeneous graph neural network. In *Proceedings of AAAI*, Vol. 37. 10816–10824.
[86] Huaxiu Yao, Chuxu Zhang, Ying Wei, Meng Jiang, Suhang Wang, Junzhou Huang, Nitesh Chawla, and Zhenhui Li. 2020. Graph few-shot learning via knowledge transfer. In *Proceedings of AAAI*, Vol. 34. 6656–6663.
[87] Jaehong Yoon, Saehoon Kim, Eunho Yang, and Sung Ju Hwang. 2020. Scalable and order-robust continual learning with additive parameter decomposition. In *International Conference on Learning Representation*.
[88] Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. 2017. Lifelong learning with dynamically expandable networks. *arXiv preprint arXiv:1708.01547* (2017).
[89] Wenchao Yu, Wei Cheng, Charu C Aggarwal, Kai Zhang, Haifeng Chen, and Wei Wang. 2018. 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*. 2672–2681.
[90] Qiao Yuan, Sheng-Uei Guan, Pin Ni, Tianlun Luo, Ka Lok Man, Prudence Wong, and Victor Chang. 2023. Continual Graph Learning: A Survey. *arXiv preprint arXiv:2301.12230* (2023).
[91] Hanqing Zeng, Muhan Zhang, Yinglong Xia, Ajitesh Srivastava, Andrey Malevich, Rajgopal Kannan, Viktor Prasanna, Long Jin, and Ren Chen. 2021. Decoupling the depth and scope of graph neural networks. *NeurIPS* 34 (2021).
[92] Peiyan Zhang, Yuchen Yan, Chaozhuo Li, Senzhang Wang, Xing Xie, Guojie Song, and Sunghun Kim. 2023. Continual Learning on Dynamic Graphs via Parameter Isolation. *arXiv preprint arXiv:2305.13825* (2023).
[93] Wentao Zhang, Ziqi Yin, Zeang Sheng, Yang Li, Wen Ouyang, Xiaosen Li, Yangyu Tao, Zhi Yang, and Bin Cui. 2022. Graph attention multi-layer perceptron. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 4560–4570.
[94] Xikun Zhang, Dongjin Song, Yixin Chen, and Dacheng Tao. 2024. Topology-aware Embedding Memory for Learning on Expanding Graphs. *arXiv preprint arXiv:2401.13200* (2024).
[95] Xikun Zhang, Dongjin Song, and Dacheng Tao. 2022. Cglb: Benchmark tasks for continual graph learning. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.
[96] Xikun Zhang, Dongjin Song, and Dacheng Tao. 2022. Sparsified Subgraph Memory for Continual Graph Representation Learning. In *2022 IEEE ICDM*. IEEE, 1335–1340.
[97] Xikun Zhang, Dongjin Song, and Dacheng Tao. 2023. Hierarchical Prototype Networks for Continual Graph Representation Learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45, 4 (2023), 4622–4636. https://doi.org/10.1109/TPAMI.2022.3186909
[98] Xikun Zhang, Dongjin Song, and Dacheng Tao. 2023. Ricci Curvature-Based Graph Sparsification for Continual Graph Representation Learning. *IEEE TNNLS* (2023).
[99] Xikun Zhang, Dongjin Song, and Dacheng Tao. 2024. Continual Learning on Graphs: Challenges, Solutions, and Opportunities. *arXiv preprint arXiv:2402.11565* (2024).
[100] Xikun Zhang, Chang Xu, and Dacheng Tao. 2020. Context aware graph convolution for skeleton-based action recognition. In *Proceedings of the IEEE/CVF CVPR*. 14333–14342.
[101] Xikun Zhang, Chang Xu, and Dacheng Tao. 2020. On dropping clusters to regularize graph convolutional neural networks. In *ECCV*.
[102] Xikun Zhang, Chang Xu, Xinmei Tian, and Dacheng Tao. 2019. Graph edge convolutional neural networks for skeleton-based action recognition. *IEEE TNNLS* 31, 8 (2019), 3047–3060.
[103] Xin Zheng, Yixin Liu, Shirui Pan, Miao Zhang, Di Jin, and Philip S Yu. 2022. Graph neural networks for graphs with heterophily: A survey. *arXiv preprint arXiv:2202.07082* (2022).
[104] Fan Zhou and Chengtai Cao. 2021. Overcoming catastrophic forgetting in graph neural networks with experience replay. In *Proceedings of AAAI*, Vol. 35. 4714–4722.
[105] Fan Zhou, Chengtai Cao, Kunpeng Zhang, Goce Trajcevski, Ting Zhong, and Ji Geng. 2019. Meta-gnn: On few-shot node classification in graph meta-learning. In *Proceedings of the 28th ACM CIKM*. 2357–2360.
[106] Lekui Zhou, Yang Yang, Xiang Ren, Fei Wu, and Yueting Zhuang. 2018. Dynamic network embedding by modeling triadic closure process. In *Proceedings of AAAI*, Vol. 32.
[107] Hao Zhu and Piotr Koniusz. 2020. Simple spectral graph convolution. In *ICLR*.
[108] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. 2020. Beyond homophily in graph neural networks: Current limitations and effective designs. *NeurIPS* 33 (2020), 7793–7804.

A THEORETICAL ANALYSIS

In this section, we give proofs and analysis of the theoretical results.

Proof of Theorem 1.1. Given a node v, the prediction is:

$$\hat{\mathbf{y}}_v = \mathrm{f}_{out}(\mathbf{e}_v; \boldsymbol{\theta}) \tag{12}$$

$\because \mathbf{e}_v = \sum_{w \in \mathbb{V}_v^{sub}} \mathbf{x}_w \cdot \pi(v, w; \hat{\mathbf{A}})$, where $\mathbb{V}_v^{sub}$ denotes the node set of the computation ego-subnetwork $\mathcal{G}_v^{sub}$, and $\hat{\mathbf{A}}$ is the adjacency matrix of $\mathcal{G}_v^{sub}$.

$\therefore$

$$\hat{\mathbf{y}}_v = \mathrm{f}_{out}\Big(\sum_{w \in \mathbb{V}_v^{sub}} \mathbf{x}_w \cdot \pi(v, w; \hat{\mathbf{A}}); \boldsymbol{\theta} \Big) \tag{13}$$

Given the label of node v (y_v), the objective function of training the model with node v is formulated as:

$$\mathcal{L}_v = l\bigg(\mathrm{f}_{out}\Big(\sum_{w \in \mathbb{V}_v^{sub}} \mathbf{x}_w \cdot \pi(v, w; \hat{\mathbf{A}}); \boldsymbol{\theta} \Big), \mathbf{y}_v \bigg), \tag{14}$$

where l could be any loss function. Since $\mathbb{V}_v^{sub}$ contains both the features of node v and its neighbors, Equation 14 can be further expanded to separate the contribution of node v and its neighbors:

$$\mathcal{L}_v = l\bigg(\mathrm{f}_{out}\Big(\underbrace{\mathbf{x}_v \cdot \pi(v, v; \hat{\mathbf{A}})}_{\text{information from node } v} + \underbrace{\sum_{w \in \mathbb{V}_v^{sub} \setminus \{v\}} \mathbf{x}_w \cdot \pi(v, w; \hat{\mathbf{A}})}_{\text{neighborhood information}}; \boldsymbol{\theta} \Big), \mathbf{y}_v \bigg), \tag{15}$$

Given an arbitrary node $q \in \mathbb{V}_v^{sub}$ but $q \neq v \in \mathbb{V}_v^{sub}$ (the adjacency matrix $\hat{\mathbf{A}}$ stays the same), we can similarly obtain the loss of training the model with node q:

$$\mathcal{L}_q = l\bigg(\mathrm{f}_{out}\Big(\underbrace{\mathbf{x}_q \cdot \pi(q, q; \hat{\mathbf{A}})}_{\text{information from node } q} + \underbrace{\sum_{w \in \mathbb{V}_q^{sub} \setminus \{q\}} \mathbf{x}_w \cdot \pi(q, w; \hat{\mathbf{A}})}_{\text{neighborhood information}}; \boldsymbol{\theta} \Big), \mathbf{y}_q \bigg). \tag{16}$$

Since $q \in \mathbb{V}_v^{sub} \setminus \{v\}$, we rewrite Equation 15 as:

$$\mathcal{L}_v = l\bigg(\mathrm{f}_{out}\Big(\underbrace{\mathbf{x}_q \cdot \pi(v, q; \hat{\mathbf{A}})}_{\text{information from node } q} + \underbrace{\sum_{w \in \mathbb{V}_v^{sub} \setminus \{q\}} \mathbf{x}_w \cdot \pi(v, w; \hat{\mathbf{A}})}_{\text{neighborhood information}}; \boldsymbol{\theta} \Big), \mathbf{y}_v \bigg), \tag{17}$$

By comparing Equation 17 and 16, we could observe the similarity in the loss of node v and q, and the difference lies in the contribution (weight $\pi(\cdot, \cdot; \hat{\mathbf{A}})$) of each node and the neighboring nodes ($\mathbb{V}_q^{sub}$ and $\mathbb{V}_v^{sub}$). □

Proof of Theorem 1.2. In this part, we choose the loss function l as cross entropy $\mathrm{CE}(\cdot, \cdot)$, which is the common choice for classification problems. In the following, we will first derive the gradient of training the PDGNNs with $(\mathbf{e}_v, y_v)$. For cross entropy, we denote the one-hot vector form label as $\mathbf{y}_v$, of which the y_v-th element is one and other entries are zero. Given the loss of a node v as shown in the Equation 14, the gradient is derived as:

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}_v = \nabla_{\boldsymbol{\theta}} \mathrm{CE}\bigg(\sum_{w \in \mathbb{V}_v^{sub}} \mathrm{f}_{out}\Big(\mathbf{x}_w \cdot \pi(v, w; \hat{\mathbf{A}}); \boldsymbol{\theta} \Big), \mathbf{y}_v \bigg) \tag{18}$$

$$= \nabla_{\boldsymbol{\theta}} \bigg(\mathbf{y}_{v,k} \cdot \log \sum_{w \in \mathbb{V}_v^{sub}} \mathrm{f}_{out}\Big(\mathbf{x}_w \cdot \pi(v, w; \hat{\mathbf{A}}); \boldsymbol{\theta} \Big)_k \bigg) \tag{19}$$

$$= \mathbf{y}_{v,k} \cdot \frac{\nabla_{\boldsymbol{\theta}} \Big(\sum_{w \in \mathbb{V}_v^{sub}} \mathrm{f}_{out}\big(\mathbf{x}_w \cdot \pi(v, w; \hat{\mathbf{A}}); \boldsymbol{\theta} \big)_k \Big)}{\sum_{w \in \mathbb{V}_v^{sub}} \mathrm{f}_{out}\Big(\mathbf{x}_w \cdot \pi(v, w; \hat{\mathbf{A}}); \boldsymbol{\theta} \Big)_k} \tag{20}$$

$$= \mathbf{y}_{v,k} \cdot \frac{\sum_{w \in \mathbb{V}_v^{sub}} \nabla_{\boldsymbol{\theta}} \mathrm{f}_{out}\Big(\mathbf{x}_w \cdot \pi(v, w; \hat{\mathbf{A}}); \boldsymbol{\theta} \Big)_k}{\sum_{w \in \mathbb{V}_v^{sub}} \mathrm{f}_{out}\Big(\mathbf{x}_w \cdot \pi(v, w; \hat{\mathbf{A}}); \boldsymbol{\theta} \Big)_k} \tag{21}$$

$$= \mathbf{y}_{v,k} \cdot \frac{\sum_{w \in \mathbb{V}_v^{sub}} \nabla_{\boldsymbol{\theta}} \mathrm{f}_{out}(\mathbf{x}_w; \boldsymbol{\theta})_k \cdot \pi(v, w; \hat{\mathbf{A}})}{\sum_{w \in \mathbb{V}_v^{sub}} \mathrm{f}_{out}\Big(\mathbf{x}_w \cdot \pi(v, w; \hat{\mathbf{A}}); \boldsymbol{\theta} \Big)_k} \tag{22}$$

$$= \frac{\sum_{w \in \mathbb{V}_v^{sub}} \mathbf{y}_{v,k} \cdot \frac{\nabla_{\boldsymbol{\theta}} \mathrm{f}_{out}(\mathbf{x}_w; \boldsymbol{\theta})_k}{\mathrm{f}_{out}(\mathbf{x}_w; \boldsymbol{\theta})_k} \cdot \mathrm{f}_{out}(\mathbf{x}_w; \boldsymbol{\theta})_k \cdot \pi(v, w; \hat{\mathbf{A}})}{\sum_{w \in \mathbb{V}_v^{sub}} \mathrm{f}_{out}\Big(\mathbf{x}_w \cdot \pi(v, w; \hat{\mathbf{A}}); \boldsymbol{\theta} \Big)_k} \tag{23}$$

$$= \frac{\sum_{w \in \mathbb{V}_v^{sub}} \nabla_{\boldsymbol{\theta}} \mathrm{CE}\big(\mathrm{f}_{out}(\mathbf{x}_w; \boldsymbol{\theta}), \mathbf{y}_{v,k} \big) \cdot \mathrm{f}_{out}(\mathbf{x}_w; \boldsymbol{\theta}) \cdot \pi(v, w; \hat{\mathbf{A}})}{\sum_{w \in \mathbb{V}_v^{sub}} \mathrm{f}_{out}\Big(\mathbf{x}_w \cdot \pi(v, w; \hat{\mathbf{A}}); \boldsymbol{\theta} \Big)} \tag{24}$$

$$= \sum_{w \in \mathbb{V}_v^{sub}} \frac{\mathrm{f}_{out}(\mathbf{x}_w; \boldsymbol{\theta}) \cdot \pi(v, w; \hat{\mathbf{A}})}{\sum_{w \in \mathbb{V}_v^{sub}} \mathrm{f}_{out}\Big(\mathbf{x}_w \cdot \pi(v, w; \hat{\mathbf{A}}); \boldsymbol{\theta} \Big)} \cdot \tag{25}$$

$$\nabla_{\boldsymbol{\theta}} \mathrm{CE}\big(\mathrm{f}_{out}(\mathbf{x}_w; \boldsymbol{\theta}), \mathbf{y}_v \big). \tag{26}$$

The loss of training $\mathrm{f}_{out}(\mathbf{x}_w; \boldsymbol{\theta})$ with pairs of feature and pseudo-label $(\mathbf{x}_w, y_v)$ of all nodes of $\mathcal{G}_v^{sub}$ is:

$$\mathcal{L}_{\mathcal{G}_v^{sub}} = \sum_{w \in \mathbb{V}_v^{sub}} \mathrm{CE}\big(\mathrm{f}_{out}(\mathbf{x}_w; \boldsymbol{\theta}), \mathbf{y}_v \big) \tag{27}$$

$$\tag{28}$$

Then, the corresponding gradient of $\mathcal{L}_{\mathcal{G}_v^{sub}}$ is :

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{G}_v^{sub}} = \sum_{w \in \mathbb{V}_v^{sub}} \nabla_{\boldsymbol{\theta}} \mathrm{CE}\big(\mathrm{f}_{out}(\mathbf{x}_w; \boldsymbol{\theta}), \mathbf{y}_v \big). \tag{29}$$

By comparing Equation 26 and 29, we can see that training PDGNNs with a topology-aware embedding $\mathbf{e}_v$ equals to training the function $\mathrm{f}_{out}(\cdot; \boldsymbol{\theta})$ on all nodes of the computation ego-subnetwork $\mathcal{G}_v^{sub}$ with a weight $\frac{\mathrm{f}_{out}(\mathbf{x}_w; \boldsymbol{\theta}) \cdot \pi(v, w; \hat{\mathbf{A}})}{\sum_{w \in \mathbb{V}_v^{sub}} \mathrm{f}_{out}\left(\mathbf{x}_w \cdot \pi(v, w; \hat{\mathbf{A}}); \boldsymbol{\theta} \right)}$ on each node to rescale the contribution dynamically. □