ECAI 2024
U. Endriss et al. (Eds.)

doi:10.3233/FAIA240692

Graph Continual Learning with Debiased Lossless Memory Replay

Chaoxi Niu[a], Guansong Pang[b,*] and Ling Chen[a]

[a]AAII, University of Technology Sydney, Sydney, Australia
[b]Singapore Management University, Singapore

Abstract. Real-life graph data often expands continually, rendering the learning of graph neural networks (GNNs) on static graph data impractical. Graph continual learning (GCL) tackles this problem by continually adapting GNNs to the expanded graph of the current task while maintaining the performance over the graph of previous tasks. Memory replay-based methods, which aim to replay data of previous tasks when learning new tasks, have been explored as one principled approach to mitigate the forgetting of the knowledge learned from the previous tasks. In this paper we extend this methodology with a novel framework, called Debiased Lossless Memory replay (DeLoMe). Unlike existing methods that *sample nodes/edges of previous graphs* to construct the memory, DeLoMe *learns lossless prototypical node representations* as the memory. The learned memory can not only preserve the graph data privacy but also capture the holistic graph information, both of which the sampling-based methods fail to achieve. Further, prior methods suffer from bias toward the current task due to the data imbalance between the classes in the memory data and the current data. A debiased GCL loss function is devised in DeLoMe to effectively alleviate this bias. Extensive experiments on four graph datasets show the effectiveness of DeLoMe under both class- and task-incremental learning settings. Code is available at https://github.com/mala-lab/DeLoMe.

1 Introduction

Due to the superior capacity to represent complex relations between samples, graph is widely used in various real-world applications [32, 41, 24] such as social networks, citation networks, and online shopping. For example, in the context of online shopping, consumers could be nodes and the edges between consumers could represent that they have purchased or rated the same product. In real-world applications, graph data are often expanded continually, e.g., new consumers and the associated connections would be constantly added to the online shopping network. Following the message propagation paradigm, graph neural networks (GNNs) [31, 30, 12, 44] have achieved remarkable success for various graph-related tasks. Despite the success, most GNNs operate on static graph data. Directly applying them to accommodate new emerging graphs would cause the GNNs to easily forget the knowledge learned from the previous data due to the large distribution difference between the historical data and the newly added data. The forgetting of the learned knowledge when learning new graphs, a.k.a. catastrophic forgetting, would result in deteriorated performance on historical graphs. A simple solution is to store all historical data and repeatedly retrain the GNNs whenever the graph is updated, but it is prohibitively expensive in terms of computational time and resources considering the large scale of the continually expanding graph. Moreover, the previous graph data would also be inaccessible in privacy-critical application scenarios when learning the newly added data.

To tackle catastrophic forgetting, many methods have been proposed and demonstrated impressive performance on Euclidean data such as images and texts [2, 33, 21, 29, 13, 11]. However, it is ineffective to directly adopt them for graph continual learning (GCL) [27, 40, 3] as graphs are non-Euclidean data that contain complex relations among a large number of nodes.

Recently, to address the unique challenges in graph data, several GCL works [7, 43, 15, 26, 28, 25, 23, 35, 22, 38, 39] have been proposed to continually adapt GNNs to the expanded graph data of the current task while maintaining the performance over the graph data of previous tasks. Generally, these methods can be roughly divided into three categories, including regularization-based methods, parameter isolation-based methods, and replay-based methods. Due to the straightforward intuitiveness and impressive effectiveness, replay-based methods [38, 39, 43] have been widely explored and achieved remarkable capacity against catastrophic forgetting in GCL. Typically, they sample and store representative data of previous tasks in a memory buffer and replay them when learning a new task. However, since only selected graph information is stored, their constructed memory for each previous task often struggles to capture the global structure information of the full graph (i.e., **holistic graph information**), limiting the power of memory replay against catastrophic forgetting. For example, ERGNN [43] stores only individual nodes via sampling methods for replaying, which ignores the rich relations among the nodes, severely degrading the effectiveness of memory replay in GCL. A straightforward solution to this issue is to store the complete neighbors and the associated edges of the selected nodes for graph data of previous tasks, but this would pose great challenges to memory storage and is infeasible under **tight memory budgets**. To preserve the topological information and alleviate the storage requirement simultaneously, recent studies propose approaches like SSM [39] to sparsify the neighborhood of the selected nodes by filtering unimportant ones. Nevertheless, all these methods focus on constructing the memory using partial graph data, as shown in Figure 1(Left), failing to preserve the holistic graph structure. Also, these memory construction methods would become inapplicable in privacy-critical applications where the replay of previous graph data can lead to **privacy leakage**, e.g., storing the original data of online

* Corresponding Author: G. Pang (gspang@smu.edu.sg).

shopping networks would divulge the purchase/rating information of consumers.

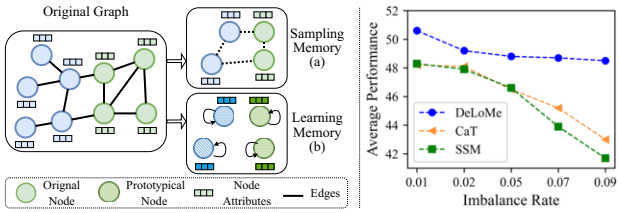


Figure 1. ***Left:*** (a) Current replay-based methods use a sampling-based memory consisting of partially sampled graph data. (b) Our approach learns to generate the memory using a lossless small graph with prototypical node representations. ***Right:*** Average accuracy (AA) of three replay methods – SSM, CaT and our DeLoMe – with increasing imbalance rates on Arxiv.

Further, since the amount of the memory data is often limited, the current graph data often dominates the training data, leading to a **data imbalance** between the classes in the memory data and that in the current graph data. When updating the GCL models with those imbalanced data, the models are biased toward the current task, amplifying the deteriorated performance on the previous tasks as the graph continually expands with a fixed memory budget (i.e., increasing imbalance rates), e.g., performance of SSM in Figure 1(Right).

To tackle these three issues, in this paper, we introduce a novel memory replay-based GCL approach, called Debiased Lossless Memory replay (**DeLoMe**). Rather than storing the original graph data, it learns a small graph consisting of *prototypical node representations as memory* so that the gradient of a randomly initialized GNN on the original large graph is lossless compared to that on the learned small graph. In doing so, the learned prototypical representations are enforced to better capture the holistic graph structure and attribute information, resulting in better performance on previous graph data, as illustrated in Figure 1(Right). This node representation-based memory also helps better preserve the privacy of the graph data, compared to the original node/edge-based memory.

To handle the aforementioned bias issue, a debiased loss function is devised in our GCL objective. This is done by calibrating the prediction logits of the classes in the memory data and the current graph data in the continual updating of our DeLoMe model, which helps largely enhance its robustness w.r.t. the class imbalance.

Our main contributions can be summarized as follows:

- We propose a novel learnable memory replay-based approach DeLoMe. Compared to the concurrent work CaT [17] that introduces a seminal memory learning-based GCL method, DeLoMe introduces an enhanced graph memory learning method and augments it with a debiased GCL objective, resulting in the first GCL framework for debiased learnable memory-based replay.
- To obtain such memory, we introduce a lossless GCL memory learning method that utilizes gradient matching to enforce a lossless compression of large graphs of previous tasks into a small set of prototypical node representations as the memory data. It enables DeLoMe to achieve not only a small memory budget requirement but also good privacy preservation of historical data.
- To mitigate the bias toward the current graph, we devise a debiased GCL objective that effectively calibrates the GNN predictions when adapting the GCL models.
- Extensive experiments on four real-world datasets show that DeLoMe learns substantially more cost-effective memory and outperforms both state-of-the-art sampling and learnable memory-based methods under both class- and task-incremental settings.

2 Related Work

2.1 Graph Continual Learning

Graph continual learning (GCL) has gained growing popularity in deep learning, and various methods have been proposed [7, 43, 15, 26, 28, 25, 23, 35, 22, 38, 39, 34], which can be divided into three categories, i.e., regularization-based, parameter isolation-based, and data replay-based methods. For example, as a regularization-based method, TWP [15] preserved the important parameters in the topological aggregation and loss minimization for previous tasks via regularization terms. Among the parameter isolation methods, HPNs [37] extracted different levels of abstract knowledge in the form of prototypes and selected different combinations of parameters for different tasks, and [35] proposed to continually expand model parameters to learn new emerging graph patterns. Differently, ERGNN [43] proposed to sample and store representative nodes of previous tasks in a memory buffer, and replayed them when learning new tasks. However, the graph structure, which plays a vital role in graph representation learning, is not considered in ERGNN. To cope with the topological information, [38] and [39] proposed the sparsification techniques to find the important neighbors of the selected nodes. Then, the important neighbors together with the edges between neighbors and representative nodes are stored in the memory buffer for replaying during the learning of new tasks. Nevertheless, all these methods focus on constructing the memory using partial graph data of previous graphs, failing to preserve the holistic graph semantics and also raising privacy concerns in privacy-critical applications. By contrast, our learned memory data helps capture the holistic graph information while effectively preserving the privacy of the previous graph data.

2.2 Graph Condensation

Graph condensation aims to compress a graph into a significantly smaller graph while preserving the holistic information of the original graph. Various graph compression methods have been proposed [9, 10, 5, 16, 42] that are based on gradient matching or distribution matching between the compressed graph and the original graph. In this work, we utilize gradient matching to compress the graphs with node features and structure in previous tasks into comprehensive prototypical node representations, which can serve as the memory for the subsequent replaying. Note that the concurrent work CaT [17] shares a common motivation with our method. However, unlike CaT which adopts a distribution matching-based method to learn the memory, we propose to use a gradient matching method that can measure the loss of condensation in a more fine-grained manner. CaT performs the GNNs training within the learned memory bank to alleviate the class imbalance problem, but it can lead to less effective utilization of the graph data of the current task. We instead devise a debiased GCL objective to calibrate the GCL predictions while avoiding the inefficient exploitation of the current graph data.

3 Preliminaries

3.1 The GCL Problem

In this paper, we focus on the node-level GCL problem. Formally, this problem can be formulated as learning a model on a sequence of graphs (tasks) $\{\mathcal{G}_1, \ldots, \mathcal{G}_T\}$ where T is the number of continual learning tasks. Each $\mathcal{G}_t = (A_t, X_t)$ is a newly emerging graph at task t, where A_t denotes the relations between nodes, X_t represents the node features, and the labels of nodes can be denoted

as Y_t. Generally, each task contains a unique set of classes, i.e., $\{Y_t \cap Y_j = \varnothing | t \neq j\}$. When learning task t, the model trained from previous tasks only has access to current data $\mathcal{G}_t$. The goal is to adapt the model to current graph $\mathcal{G}_t$ while maintaining the classification performance on the previous graphs $\{\mathcal{G}_1, \ldots, \mathcal{G}_{t-1}\}$.

3.2 Class & Task Incremental Settings of GCL

Depending on whether the task indicator is provided at the testing stage, GCL can be further divided into two settings: Class-Incremental Learning (CIL) and Task-Incremental Learning (TIL). Assume that each task in $\{\mathcal{G}_1, \ldots, \mathcal{G}_T\}$ has the same number of C classes and we use C_t to represent the unique set of classes in $\mathcal{G}_t$. In CIL, after learning all the T tasks, the model is required to classify each test instance into one of all the learned $T \times C$ classes. While under the TIL setting, the task indicator t is also provided with the test instance. Thus, the model is only required to assign a class to the test instance within C_t of task t. Compared to TIL, CIL is more practical yet more challenging as the label prediction space contains all the learned classes so far. In this paper, we evaluate the proposed method under both settings to demonstrate its effectiveness.

3.3 Memory Replay in Graph Continual Learning

In GCL, a GNN model is sequentially trained across the task sequence $\{\mathcal{G}_1, \ldots, \mathcal{G}_T\}$. Typically, the model only has access to $\mathcal{G}_t = (A_t, X_t)$ at task t. To continually accommodate the new graph and alleviate the catastrophic forgetting, memory replay-based methods store representative data of previous $t-1$ tasks into a memory buffer $\mathcal{B}_t$. Then, the memory data are replayed with the data of task t to fit the new graph data and maintain the knowledge of previous tasks simultaneously. Therefore, the training objective of memory replay at task t can be formulated as follows:

$$\mathcal{L} = \underbrace{\ell(f_\theta(\mathcal{G}_t), Y_t)}_{\text{current task loss}} + \lambda \underbrace{\ell(f_\theta(\mathcal{B}_t), Y_{\mathcal{B}_t})}_{\text{memory loss}}, \tag{1}$$

where $f_\theta(\cdot)$ is the GNN model parameterized by θ, $Y_{\mathcal{B}_t}$ denote the labels of nodes in the memory buffer $\mathcal{B}_t$ (i.e., all class labels encountered in the previous $t-1$ tasks), $\ell(\cdot)$ denotes a loss function, i.e., cross-entropy loss, and λ is a hyperparameter to control the importance of the memory loss.

The memory buffer plays an important role in maintaining previously learned knowledge and different memory construction methods have been proposed. For example, ERGNN [43] sampled the representative nodes of the previous tasks as the memory. However, the rich topological information is neglected in ERGNN. Other approaches like SSM [39] aim to utilize the structure information by sparsifying the neighbors of the sampled nodes based on their topological importance. Then, the important neighborhood structures together with the sampled nodes are used to construct the memory. Despite the success achieved by these methods, the memory buffer constructed with partial graph data fails to preserve the holistic semantics of the original graph and it may lead to privacy leakage issue when the sampled nodes/edges are sensitive data. Further, the memory budget is typically kept significantly smaller than the current graph data, so it can lead to class imbalance between the memory data and the current graph data. Our method DeLoMe is proposed to tackle these three issues.

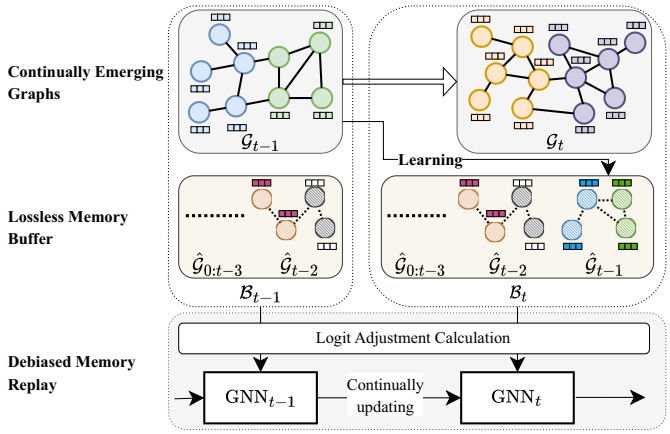


Figure 2. Overview of DeLoMe. We take two consecutive tasks ($\mathcal{G}_{t-1}$ and $\mathcal{G}_t$) as an example, where GNN_{t-1} and GNN_t represent the GNN model trained after task $t-1$ and t respectively. At task $t-1$, we learn prototypical node representation-based memory $\hat{\mathcal{G}}_{t-1}$ for $\mathcal{G}_{t-1}$ and add it to the memory buffer via $\mathcal{B}_t = \mathcal{B}_{t-1} \cup \hat{\mathcal{G}}_{t-1}$. At task t, the memory buffer $\mathcal{B}_t$ is replayed with the current graph data $\mathcal{G}_t$ to train the model GNN_t using our debiased GCL objective. The process is repeated until all the tasks are learned.

4 Debiased Lossless Memory Replay

4.1 Lossless Memory Learning

4.1.1 Key intuition

Instead of using partial graph data as memory replay data, we propose to learn a small set of prototypical node representatives as the memory data which can holistically represent the original graph structure and attributes. Taking the task $t-1$ as an example, given $\mathcal{G}_{t-1} = (A_{t-1}, X_{t-1})$ with the label set Y_{t-1}, we aim to learn a *compressed graph* $\hat{\mathcal{G}}_{t-1} = (I, \hat{X}_{t-1})$ associated with label set $\hat{Y}_{t-1}$ as our memory data, where the fixed identity matrix I represents the structure of $\hat{\mathcal{G}}_{t-1}$ with learned prototypical nodes. Note that the size of $\hat{X}_{t-1}$ is constrained by the memory budget and is significantly smaller than X_{t-1}.

Since the learned $\hat{\mathcal{G}}_{t-1}$ captures the holistic semantics of the original graph $\mathcal{G}_{t-1}$ at task $t-1$, we can expect that a GNN model trained on $\hat{\mathcal{G}}_{t-1}$ would achieve comparable performance to that trained on $\mathcal{G}_{t-1}$. Following this idea, the learning objective of $\hat{\mathcal{G}}_{t-1}$ can be formulated as follows:

$$\min_{\hat{\mathcal{G}}_{t-1}} \ell(f_{\hat{\theta}}(\mathcal{G}_{t-1}), Y_{t-1}), \ \text{s.t.} \ \hat{\theta} = \arg\min_{\theta} \ell(f_\theta(\hat{\mathcal{G}}_{t-1}), \hat{Y}_{t-1}), \tag{2}$$

where $\hat{\theta}$ denotes the parameters of the graph neural network $f(\cdot)$ trained on $\hat{\mathcal{G}}_{t-1}$. Due to the nested loop optimization of Eq. (2), directly solving the above objective would be prohibitively expensive. To address this challenge, one-step gradient matching [9] is proposed, which aims to match the gradient of the same model with regard to the real data and the learned data at the first training epoch. Inspired by this, the optimization objective Eq. (2) can be transformed into a lossless compression as follows:

$$\min_{\hat{\mathcal{G}}_{t-1}} d(\nabla_\theta \ell(f_\theta(\mathcal{G}_{t-1}), Y_{t-1}), \nabla_\theta \ell(f_\theta(\hat{\mathcal{G}}_{t-1}), \hat{Y}_{t-1})), \tag{3}$$

where $d(\cdot)$ is a distance function to measure the difference between the two gradients. Moreover, to get more generalized $\hat{\mathcal{G}}_{t-1}$ without fitting to a specific model initialization, we aim to devise an objective

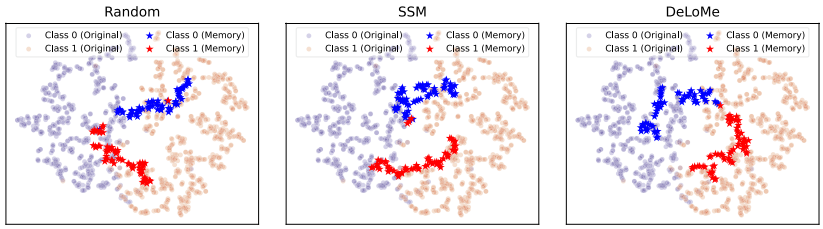


Figure 3. Visualization of node embeddings of the original graph and the memories obtained by different methods on this graph.

of Eq. (3) that can be minimized under different random initializations of the $f_\theta(\cdot)$. Thus, the final objective to learn $\hat{\mathcal{G}}_{t-1}$ is defined as follows:

$$\min_{\hat{\mathcal{G}}_{t-1}} \sum_{\theta_p \sim \Theta} d(\nabla_{\theta_p} \ell(f_{\theta_p}(\mathcal{G}_{t-1}), Y_{t-1}), \nabla_{\theta_p} \ell(f_{\theta_p}(\hat{\mathcal{G}}_{t-1}), \hat{Y}_{t-1})), \tag{4}$$

where θ_p is a random instantiation of the parameter space Θ.

By optimizing Eq. (4), we obtain the compressed graph $\hat{\mathcal{G}}_{t-1}$, in which a small set of prototypical nodes are learned for each class in $\mathcal{G}_{t-1}$. To achieve the lossless compression, these prototypical nodes are enforced to capture diverse important class-level structure and attribute information. $\hat{\mathcal{G}}_{t-1}$ is then used to update the memory buffer $\mathcal{B}_{t-1}$, i.e., $\mathcal{B}_t = \mathcal{B}_{t-1} \cup \hat{\mathcal{G}}_{t-1}$. Then, $\mathcal{B}_t$ is replayed with the graph $\mathcal{G}_t$ when learning the next task t (see Figure 2).

4.1.2 *The learning algorithm*

To obtain the memory $\hat{\mathcal{G}}_{t-1} = (I, \hat{X}_{t-1})$ with $\hat{Y}_{t-1}$ of task $t-1$, we need to learn the node feature $\hat{X}_{t-1}$ and label set $\hat{Y}_{t-1}$. Since $\hat{Y}_{t-1}$ represents the node label and is discrete, $\hat{Y}_{t-1}$ is fixed as the same classes as the original label set Y_{t-1}. Therefore, we only need to learn $\hat{X}_{t-1}$ for task $t-1$. To accelerate the learning process, let b be the memory budget for each class, $\hat{X}_{t-1}$ is initialized as the features of randomly selected b nodes from each class in $\mathcal{G}_{t-1}$. To further reduce the computation cost, we sample a fixed number of neighbors for each node in $\mathcal{G}_{t-1}$ at each hop, and adopt the mini-batch training strategy. In addition, the gradient matching and learning of $\hat{X}$ is performed on each class separately. Specifically, for a given class c in Y_{t-1}, a batch of nodes belonging to class c is randomly sampled from $\mathcal{G}_{t-1}$ together with the associated neighborhoods, which is denoted as $\mathcal{G}^c_{t-1} = (A^c_{t-1}, X^c_{t-1})$. Meanwhile, we get the corresponding nodes of class c from $\hat{\mathcal{G}}_{t-1}$ and denote it as $\hat{\mathcal{G}}^c_{t-1} = (I, \hat{X}^c_{t-1})$. Then, the sampled $\mathcal{G}^c_{t-1}$ and $\hat{\mathcal{G}}^c_{t-1}$ are fed into the same GNN model to calculate the gradient matching loss. Finally, $\hat{X}^c_{t-1}$ is optimized via gradient descent to minimize the graph matching loss. Note that the graph neural network $f_\theta(\cdot)$ is not updated during the learning process and we adopt different initializations of $f_\theta(\cdot)$ to learn a generalized $\hat{\mathcal{G}}_{t-1}$. By imitating the training trajectory of the original graph $\mathcal{G}_{t-1}$, $\hat{\mathcal{G}}_{t-1} = (I, \hat{X}_{t-1})$ is enforced to capture the holistic semantics of $\mathcal{G}_{t-1}$ and the global structure information is implicitly incorporated into $\hat{X}_{t-1}$. The algorithm of lossless memory learning at the task $t-1$ in Algorithm 1.

As illustrated in Figure 3 where we visualize the node embeddings of the memories constructed by two sampling methods (random node sampling and SSM) and DeLoMe, the prototypical node representations in the memory learned by DeLoMe can better preserve the distribution of the nodes of different classes in the original graph, compared to the other two methods. This indicates the better capability of DeLoMe in capturing more holistic graph semantics.

Algorithm 1: Memory Learning for graph data of task $t-1$

Input: Graph data $\mathcal{G}_{t-1} = (A_{t-1}, X_{t-1})$ with label Y_{t-1}, memory budget b, graph neural network $f_\theta(\cdot)$, learning rate η, and the number of epochs E.
Output: Memory data $\hat{\mathcal{G}}_{t-1} = (I, \hat{X}_{t-1})$ with label $\hat{Y}_{t-1}$
Set $\hat{Y}_{t-1}$ to fixed class values as in Y_{t-1}, initialize $\hat{X}$ by randomly selecting b nodes from each class in $\mathcal{G}_{t-1}$.
for $e = 1, \ldots, E$ **do**
  Initialize graph neural network parameter θ_p from Θ
  for $c = 1, \ldots, C$ **do**
    Sample $\mathcal{G}^c_{t-1} = (A^c_{t-1}, X^c_{t-1})$ and Y^c_{t-1} from $\mathcal{G}_{t-1}$
    Sample $\hat{\mathcal{G}}^c_{t-1} = (I, \hat{X}^c_{t-1})$ and $\hat{Y}^c_{t-1}$ from $\hat{\mathcal{G}}_{t-1}$
    Compute gradient: $G^c_{t-1} = \nabla_{\theta_p} \ell(f_{\theta_p}(\mathcal{G}^c_{t-1}), Y^c_{t-1})$
    Compute gradient: $\hat{G}^c_{t-1} = \nabla_{\theta_p} \ell(f_{\theta_p}(\hat{\mathcal{G}}^c_{t-1}), \hat{Y}^c_{t-1})$
    Update $\hat{X} = \hat{X} - \eta \nabla_{\hat{X}} D(G^c_{t-1}, \hat{G}^c_{t-1})$
  end for
end for

4.1.3 *Time complexity analysis*

We analyze the time complexity of obtaining $\hat{\mathcal{G}}_{t-1} = (I, \hat{X}_{t-1})$ at task $t-1$. As mentioned above, the learning process is conducted for each class $\mathcal{G}^c_{t-1} = (A^c_{t-1}, X^c_{t-1})$ separately. We take a two-layer SGC [30] as the GNN model for gradient matching and denote the number of nodes and edges in $\mathcal{G}^c_{t-1}$ as N^n_c and N^e_c respectively. The dimension of the node features is denoted as F. For the two-layer SGC, the time complexity of forward and backward propagation with regard to $\mathcal{G}^c_{t-1}$ and $\hat{\mathcal{G}}^c_{t-1}$ are $\mathcal{O}(4N^e_c F + 2N^n_c FC))$ and $\mathcal{O}(2bFC)$ respectively. Given the training epochs E, the time complexity for class c is $\mathcal{O}(4N^e_c EF + 2N^n_c EFC + 2bEFC)$. Then, the overall time complexity of learning $\hat{\mathcal{G}}_{t-1}$ is $\mathcal{O}(\sum_{c=1}^{C}(4N^e_c EF + 2N^n_c EFC + 2bEFC))$. Since we adopt the minibatch training strategy and sample a fixed number of neighbors for each node at each hop, the numbers of N^n_c and N^e_c are typically small and do not induce high computation. Besides, the learning process can be implemented in parallel in practice to further reduce the learning time.

4.2 *Debiased Memory Replay*

Although the learned memory data are lossless compared to previous graphs, there is a data imbalance between the classes in the new graph $\mathcal{G}_t$ and that in the memory buffer $\mathcal{B}_t$. This is because the memory budget b of each class is typically much smaller than the number of nodes belonging to new classes in $\mathcal{G}_t$. This class imbalance would increase, when the graph evolves with more current graph data or a tighter memory budget is given, amplifying the degraded performance for GCL. To tackle this problem, we propose a debiased memory replay method that adjusts the prediction logits of the classes in the memory data and the current graph data based on the class label frequencies during the memory replay. Specifically, at task t, we have

the memory buffer $\mathcal{B}_t = \{\hat{\mathcal{G}}_1, \ldots, \hat{\mathcal{G}}_{t-1}\}$, and the vanilla objective of memory replay in Eq. (1) can be explicitly formulated as:

$$\mathcal{L} = \ell(H_t, Y_t) + \lambda \sum_{j=1}^{t-1} \ell(\hat{H}_j, \hat{Y}_j)\,, \tag{5}$$

where H_t and $\hat{H}_j$ are the prediction logits of $f_\theta(\cdot)$ for the nodes in $\mathcal{G}_t$ and $\hat{\mathcal{G}}_j$ respectively. Directly minimizing Eq. (5) would make the model biased toward the current task as the current graph data $\mathcal{G}_t$ dominates the training data. We address this problem by calibrating the logits H_t and $\hat{H}_j$ based on their label frequencies. Given the memory budget b, the calibration magnitude for each class in the memory buffer is equal and can be defined as:

$$\Pi_{\mathcal{B}_t} = \tau \log \frac{b}{|Y_t| + (t-1)bC}\,, \tag{6}$$

where we assume each task contains the same C classes, τ is a scaling hyperparameter, and $|Y_t|$ returns the number of samples in Y_t. For a class c in the current graph $\mathcal{G}_t$, the calibration magnitude is defined as:

$$\Pi_t^c = \tau \log \frac{|Y_t^c|}{|Y_t| + (t-1)bC}\,, \tag{7}$$

where $|Y_t^c|$ denotes the number of training samples of class c in $\mathcal{G}_t$. By incorporating these two calibrations into Eq. (5), our debiased GCL training loss is as follows:

$$\mathcal{L} = \sum_{c=1}^{C} \ell(H_t^c + \Pi_t^c, Y_t^c) + \sum_{j=1}^{i-1} \ell(\hat{H}_j + \Pi_{\mathcal{B}}, \hat{Y}_j)\,, \tag{8}$$

where we discard the weight parameter λ to simplify the parameter selection. Compared to Eq. (5), Eq. (8) augments the softmax cross-entropy with a pairwise margin based on label frequencies [20, 4]. In this way, the predictions for dominant classes in the current graph do not overwhelm those for tail classes in the memory data, thus reducing the bias toward the dominant classes in $\mathcal{G}_t$. The detailed training steps of DeLoMe are presented in Algorithm 2.

Algorithm 2: DeLoMe

Input: A sequence of graph learning tasks: $\{\mathcal{G}_1, \ldots, \mathcal{G}_T\}$ and an empty memory buffer $\mathcal{B}_1$ with a budget b for each class.
Output: A continually learned graph neural network $f_\theta(\cdot)$
Initialize $f_\theta(\cdot)$.
for $t = 1, \ldots, T$ **do**
  if $t > 1$ **then**
    Update memory buffer $\mathcal{B}_{t-1}$ with $\hat{\mathcal{G}}_{t-1}$ (Eq.(4)), i.e., $\mathcal{B}_t = \mathcal{B}_{t-1} \cup \hat{\mathcal{G}}_{t-1}$
    Calculate the logit adjustments via Eq.(6) and Eq.(7)
  end if
  Update $f_\theta(\cdot)$ by minimizing training objective Eq.(8)
  Obtain $\hat{\mathcal{G}}_t$ with Algorithm 1
end for

5 Experiments

5.1 Datasets

Following the GCL benchmark [36], four public graph datasets are employed, i.e., CoraFull [19], Arxiv [8], Reddit [6] and Products [8]. Specifically, CoraFull and Arxiv are citation networks, Reddit is constructed from Reddit posts, and Products is a co-purchasing network from Amazon. For all datasets, each task is set to contain only two classes [36]. Besides, for each class, the proportions of training, validation and testing are set to be 0.6, 0.2 and 0.2 respectively. The statistics of these datasets are summarized in the Table 1.

Table 1. Statistics of the graph datasets.

Datasets	CoraFull	Arxiv	Reddit	Products
# nodes	19,793	169,343	227,853	2,449,028
# edges	130,622	1,166,243	114,615,892	61,859,036
# classes	70	40	40	46
# tasks	35	20	20	23
# Avg. nodes per task	660	8,467	11,393	122,451
# Avg. edges per task	4,354	58,312	5,730,794	2,689,523

5.2 Competing Models

Two categories of state-of-the-art (SOTA) continual learning methods are employed for comparison. The first category contains traditional continual learning methods, i.e., EWC [13], LwF [14], GEM [18] and MAS [1]. The second category includes four SOTA GCL methods: ERGNN [43], TWP [15], HPNs [37], SSM [38], SEM [39] and CaT [17]. In addition, we include two other methods: Joint and Fine-tune. The Joint method is an oracle model that can see all graphs at all times and performs GCL on the full graphs of all tasks, while Fine-tune is a baseline that simply fine-tunes the learned model from previous tasks without continual learning techniques.

5.3 Evaluation Metrics

Average accuracy (AA) and average forgetting (AF) are adopted to evaluate the model performance. Specifically, AA and AF are calculated from the accuracy matrix $M \in \mathbb{R}^{T\times T}$, where T is the number of the tasks. The entry $M_{tj}(t \geqslant j)$ denotes the classification accuracy on task j after the model is optimized on task t. After learning all the T tasks, the overall AA and AF can be calculated as follows:

$$\text{AA} = \frac{\sum_{j=1}^{T} M_{Tj}}{T}\,, \quad \text{AF} = \frac{\sum_{j=1}^{T-1}(M_{Tj} - M_{jj})}{T-1}\,. \tag{9}$$

To sum up, AA evaluates the average performance of the model on all the learned tasks after learning the current task, and AF describes how the performance of previous tasks is affected by the current task. For both AA and AF, the higher value denotes the better GCL performance.

5.4 Implementation Details

To have a fair comparison, we implement the proposed method with the GCL benchmark [36]. More specifically, we adopt the two-layer SGC [30] as the backbone model with the same hyper-parameters following [39]. The memory budget is also set as the same in [39], i.e., 60 per class for the CoraFull dataset and 400 per class for the other datasets. For the lossless memory learning module, we also use a two-layer SGC as the GNN model to match the gradients of both the original graph and the compressed graph, and the gradient divergence is calculated based on the mean square distance. For each dataset, we report the average performance with standard deviations after 5 runs with different seeds under both task incremental and class incremental settings. More details are in Supplementary Materials[1].

[1] https://github.com/mala-lab/DeLoMe

Table 2. Results (mean±std) under the class-incremental learning setting on four datasets. Fine-tune and Joint are shown to respectively serve as approximated lower bound and upper bound performance. The best performance achieved by continual learning methods on each dataset is highlighted in bold. "↑" denotes the higher value represents better performance.

Methods	CoraFull		Arxiv		Reddit		Products	
	AA/%↑	AF/%↑	AA/%↑	AF/%↑	AA/%↑	AF/%↑	AA/%↑	AF/%↑
Fine-tune	3.5±0.5	-95.2±0.5	4.9±0.0	-89.7±0.4	5.9±1.2	-97.9±3.3	7.6±0.7	-88.7±0.8
Joint	81.2±0.4	-	51.3±0.5	-	97.1±0.1	-	71.5±0.1	-
EWC	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	6.5±1.5	-92.3±1.5	4.8±0.4	-72.2±4.1	9.2±14.5	-23.1±28.2	16.7±4.8	-57.0±31.9
GEM	8.4±1.1	-88.4±1.4	4.9±0.0	-89.8±0.3	11.5±5.5	-92.4±5.9	4.5±1.3	-94.7±0.4
LwF	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
TWP	62.6±2.2	-30.6±4.3	6.7±1.5	-50.6±13.2	8.0±5.2	-18.8±9.0	14.1±4.0	-11.4±2.0
ERGNN	34.5±4.4	-61.6±4.3	21.5±5.4	-70.0±5.5	82.7±0.4	-17.3±0.4	48.3±1.2	-45.7±1.3
SSM-uniform	73.0±0.3	-14.8±0.5	47.1±0.5	-11.7±1.5	94.3±0.1	-1.4±0.1	62.0±1.6	-9.9±1.3
SSM-degree	75.4±0.1	-9.7±0.0	48.3±0.5	-10.7±0.3	94.4±0.0	-1.3±0.0	63.3±0.1	-9.6±0.3
SEM-curvature	77.7±0.8	-10.0±1.2	49.9±0.6	-8.4±1.3	96.3±0.1	-0.6±0.1	65.1±1.0	-9.5±0.8
CaT	80.4±0.5	-5.3±0.4	48.2±0.4	-12.6±0.7	97.3±0.1	-0.4±0.0	**70.3±0.9**	**-4.5±0.8**
DeLoMe (Ours)	**81.0±0.2**	**-3.3±0.3**	**50.6±0.3**	**5.1±0.4**	**97.4±0.1**	**-0.1±0.1**	67.5±0.7	-17.3±0.3

Table 3. Full results (mean±std) under the task-incremental learning setting.

Methods	CoraFull		Arxiv		Reddit		Products	
	AA/%↑	AF/%↑	AA/%↑	AF/%↑	AA/%↑	AF/%↑	AA/%↑	AF/%↑
Fine-Tune	56.0±4.2	-41.0±4.5	56.2±2.6	-36.2±2.6	79.5±24.2	-11.7±4.8	64.4±3.8	-31.1±4.4
Joint	95.5±0.2	-	90.3±0.4	-	99.5±0.0	-	95.3±0.8	-
EWC	89.8±1.0	-5.1±0.5	71.5±0.6	-0.9±0.6	83.9±15.1	-2.0±1.5	87.0±1.4	-1.7±1.2
MAS	92.2±0.9	-3.7±1.3	72.7±2.6	-18.5±2.5	61.1±7.1	-0.5±1.0	80.6±4.3	-13.7±3.7
GEM	91.5±0.5	-1.9±0.9	81.1±1.7	-4.0±1.8	98.9±0.1	-0.5±0.1	87.7±1.8	-7.0±2.0
LwF	93.8±0.1	-0.4±0.1	71.1±3.2	-1.5±0.8	98.6±0.1	-0.0±0.0	86.3±0.2	-0.5±0.1
TWP	94.3±0.9	-1.6±0.4	89.4±0.4	0.0±0.3	78.0±18.5	-0.2±0.4	81.8±3.3	-0.3±0.8
HPNs	-	-	85.8±0.7	**0.6±0.9**	-	-	80.1±0.8	2.9±1.0
ERGNN	86.3±1.0	-9.2±0.9	86.4±0.3	0.5±0.6	97.4±0.2	**4.7±0.1**	86.4±0.0	**11.7±0.0**
SSM-uniform	95.3±0.5	0.2±0.5	88.5±0.6	-1.3±0.5	99.2±0.0	-0.2±0.0	93.1±0.8	-1.8±0.3
SSM-degree	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	**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
CaT	95.0±0.2	1.6±0.7	90.3±0.3	0.3±0.4	99.2±0.0	0.0±0.0	94.7±0.1	-0.0±0.1
DeLoMe (Ours)	95.4±0.1	**2.0±0.6**	**90.4±0.3**	-1.1±0.2	**99.4±0.0**	-0.1±0.0	**94.8±0.1**	-2.2±0.2

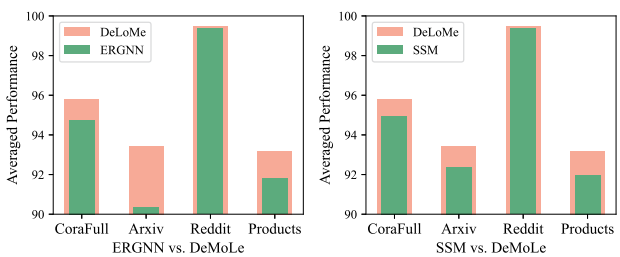


Figure 4. Average accuracy (AA) of DeLoMe against SOTA sampling-based memory construction methods on all tasks.

5.5 Main Results

5.5.1 Results under class-incremental learning (CIL)

The results of all methods under the CIL setting are shown in Table 2. Note that the results of baselines are taken from the paper [39] since we adopt the same GCL benchmark [36][2]. From the table, we can draw the following observations: (1) Directly fine-tuning the learned model from previous tasks on the current task data leads to serious performance degradation because the knowledge of previous tasks could be easily overwritten by the new tasks. (2) Continual learning methods proposed for Euclidean data generally do not achieve satisfactory performance for GCL, which verifies the fact that the unique graph properties should be taken into consideration for GCL. (3) Replay-based graph continual learning methods achieve much better performance than other baselines. Among them, SSM and SEM outperform ERGNN on all datasets. The reason could be attributed to that SSM and SEM preserve the topological information for the historical graph data in the memory while ERGNN only stores the individual nodes. (4) Differently, CaT and DeLoMe propose to learn the memory and capture the semantics of the original graph. The performance gain of CaT and DeLoMe over SSM and SEM demonstrates that the learned memory buffer is more informative (e.g., in capturing holistic graph information) and enhances the power of replaying memory. (5) Our method DeLoMe achieves new SOTA performance in nearly all cases. Its performance can even match or outperform the ideal method Joint on the CoraFull and Reddit datasets. Note that on Products DeLoMe outperforms all methods except CaT. This may be attributed to that the node class frequency information in Products does not help accurately capture the imbalance bias due to the possible presence of less informative nodes in this largest dataset, rendering our debiased component less effective.

5.5.2 Results under task-incremental learning (TIL)

In Table 3, we report the results under the TIL setting, in which a SOTA TIL method HPNs is also included for comparison. From the table, we can first see that all the methods achieve much better performance under the TIL setting. This is because the availability of task indicators during inference makes TIL much easier than CIL. Despite

[2] https://github.com/QueuQ/CGLB/tree/master

the performance of our DeLoMe being slightly inferior to SEM [39] on the CoraFull dataset, DeLoMe achieves comparable performance with the Oracle model, Joint. For the other three datasets, DeLoMe achieves the best performance and even outperforms Joint on Arxiv dataset, which verifies the advantages of our two components in overcoming the forgetting and class imbalance problems.

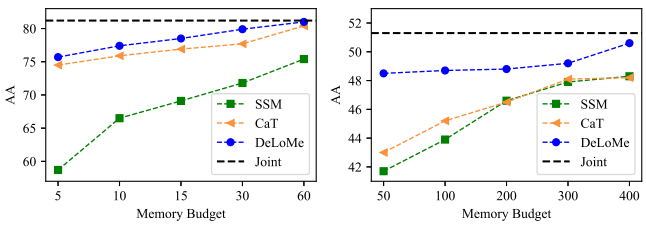
5.6 Cost-effective Learning of Expressive Memory

5.6.1 Expressiveness of sampling- vs. learning-based memory

In this subsection, we evaluate the expressiveness of the memory constructed by our learning-based method against two SOTA sampling-based methods, ERGNN and SSM. To evaluate the memory expressiveness, for each task, we train a GNN model with the memory data and calculate the node classification accuracy on the test set of the original graph. We follow the same experimental setting in the above GCL experiments and report the average classification accuracy of all tasks in Figure 4. It is clear that our learning-based method achieves much better performance than both sampling-based methods, achieving improvement by up to 3% in AA. This demonstrates the superiority of capturing the semantics of the original graph when constructing memory and explains the performance gain over the sampling-based methods in the GCL experiments.

5.6.2 Memory budget efficiency

We evaluate the performance of the proposed method with different memory budgets, with two best-competing methods – SSM and CaT – and the oracle model Joint as the baselines. Due to the page limits, we report the results on CoraFull and Arxiv under CIL. The memory budgets vary in $\{5, 10, 15, 30, 60\}$ for CoraFull and $\{50, 100, 200, 300, 400\}$ for Arxiv. The results are shown in Figure 5 which demonstrates that the learning-based methods (DeLoMe and CaT) are more effective than the sampling-based method with tight memory budgets. Compared to SSM and CaT, DeLoMe performs much more stably and achieves consistently better performance with varying budgets, especially on the Arxiv dataset. These results reinforce our empirical evidence on the effectiveness of DeLoMe in constructing memory and handling the class imbalance problem.



(a) CoraFull (b) Arxiv

Figure 5. AA results with different memory budgets on CoraFull and Arxiv under the class-incremental learning.

5.6.3 Computational efficiency

We further investigate the memory construction and inference times of DeLoMe and employ SSM and CaT for comparison. Specifically, we report the average memory construction time per task and the overall inference time of each method on the largest dataset, Products. From the results in Table 4, we can see that CaT and DeLoMe require almost the same time for memory construction while SSM is much faster than the two learning-based methods. This is because CaT and DeLoMe involve model optimization to enhance the memory to capture the holistic semantics of the original graph while SSM employs the parameter-free sampling strategy. This also explains the superiority of CaT and DeLoMe over SSM in AA and AF in Tables 2 and 3. In terms of the inference time, the three methods are nearly the same since the memory construction only happens in the training stage and the test setting remains the same for all methods.

Table 4. Time (second) of different stages on Products.

Stage	SSM	CaT	DeloMe
Memory Construction	0.53	28.94	28.37
Inference	1.73	1.75	1.71

5.7 Ablation Study

We evaluate the contribution of two key components in DeLoMe, i.e., lossless memory learning and debiased GCL learning. Specifically, we derive four variants. When the lossless memory learning is not exploited, we employ the sampling strategy to construct the memory as in ERGNN. Without loss of generality, we conduct the experiments on the CoraFull and Arxiv datasets under the CIL setting. The results of different variants are shown in the Table 5. From the table, we can see that adding either of the two components contributes to a significant improvement compared to the variant that does not use both components. These showcase the importance of both components, as well as their effectiveness in respectively addressing the memory expressiveness and data imbalance problems. In general, having a more expressive memory helps achieve larger improvement than tackling the data imbalance problem. Nevertheless, with our biased GCL objective, DeLoMe can achieve further large improvement over using our expressive memory learning component alone.

Table 5. Ablation study of the two components in DeLoMe.

Lossless Memory	Debiased Learning	CoraFull		Arixv	
		AA↑	AF↑	AA↑	AF↑
×	×	36.9	-59.0	24.6	-66.1
×	✓	50.2	-40.6	33.9	-31.1
✓	×	78.5	-9.3	47.9	-15.8
✓	✓	81.0	-3.3	50.6	5.1

6 Conclusion

This paper proposes a novel memory replay-based GCL method DeLoMe. Traditional replay-based graph continual learning methods typically construct the memory of the previous task using partial graph data, failing to preserve the holistic semantics of the original graph at each task. To tackle this issue, we learn compressed prototypical node representations as the memory by a gradient matching approach. In this way, the learned representations capture the holistic graph structure and attribute information. Besides, the learned representations help preserve the privacy of the graph data when replaying. To overcome the class imbalance problem between the learned memory and the new-coming graph, we further proposed a debiased memory replay objective by calibrating the prediction logits of the classes in both memory data and the current task based on the label frequencies. Extensive experiments on four datasets demonstrate the effectiveness of the proposed method under both class- and task-incremental learning settings of GCL.

Acknowledgements

In this work, the participation of C. Niu and L. Chen is supported by Australian Research Council under Grant DP210101347.

References

[1] R. Aljundi, F. Babiloni, M. Elhoseiny, M. Rohrbach, and T. Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *Proceedings of the European conference on computer vision (ECCV)*, pages 139–154, 2018.

[2] M. Farajtabar, N. Azizan, A. Mott, and A. Li. Orthogonal gradient descent for continual learning. In *International Conference on Artificial Intelligence and Statistics*, pages 3762–3773. PMLR, 2020.

[3] F. G. Febrinanto, F. Xia, K. Moore, C. Thapa, and C. Aggarwal. Graph lifelong learning: A survey. *IEEE Computational Intelligence Magazine*, 18(1):32–51, 2023.

[4] L. Galke, I. Vagliano, B. Franke, T. Zielke, M. Hoffmann, and A. Scherp. Lifelong learning on evolving graphs under the constraints of imbalanced classes and new classes. *Neural Networks*, 164:156–176, 2023.

[5] X. Gao, T. Chen, Y. Zang, W. Zhang, Q. V. H. Nguyen, K. Zheng, and H. Yin. Graph condensation for inductive node representation learning. *arXiv preprint arXiv:2307.15967*, 2023.

[6] W. L. Hamilton, R. Ying, and J. Leskovec. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017.

[7] B. He, X. He, Y. Zhang, R. Tang, and C. Ma. Dynamically expandable graph convolution for streaming recommendation. In *Proceedings of the ACM Web Conference 2023*, pages 1457–1467, 2023.

[8] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133, 2020.

[9] W. Jin, X. Tang, H. Jiang, Z. Li, D. Zhang, J. Tang, and B. Yin. Condensing graphs via one-step gradient matching. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 720–730, 2022.

[10] W. Jin, L. Zhao, S. Zhang, Y. Liu, J. Tang, and N. Shah. Graph condensation for graph neural networks. In *International Conference on Learning Representations*, 2022.

[11] Z. Ke and B. Liu. Continual learning of natural language processing tasks: A survey. *arXiv preprint arXiv:2211.12701*, 2022.

[12] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2016.

[13] 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*, 114(13):3521–3526, 2017.

[14] Z. Li and D. Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

[15] H. Liu, Y. Yang, and X. Wang. Overcoming catastrophic forgetting in graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 8653–8661, 2021.

[16] M. Liu, S. Li, X. Chen, and L. Song. Graph condensation via receptive field distribution matching. *arXiv preprint arXiv:2206.13697*, 2022.

[17] Y. Liu, R. Qiu, and Z. Huang. Cat: Balanced continual graph learning with graph condensation. *arXiv preprint arXiv:2309.09455*, 2023.

[18] D. Lopez-Paz and M. Ranzato. Gradient episodic memory for continual learning. *Advances in neural information processing systems*, 30, 2017.

[19] A. K. McCallum, K. Nigam, J. Rennie, and K. Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3:127–163, 2000.

[20] A. K. Menon, S. Jayasumana, A. S. Rawat, H. Jain, A. Veit, and S. Kumar. Long-tail learning via logit adjustment. In *International Conference on Learning Representations*, 2021.

[21] O. Ostapenko, M. Puscas, T. Klein, P. Jahnichen, and M. Nabi. Learning to remember: A synaptic plasticity driven framework for continual learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11321–11329, 2019.

[22] M. Perini, G. Ramponi, P. Carbone, and V. Kalavri. Learning on streaming graphs with experience replay. In *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*, pages 470–478, 2022.

[23] A. Rakaraddi, L. Siew Kei, M. Pratama, and M. De Carvalho. Reinforced continual learning for graphs. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pages 1666–1674, 2022.

[24] Y. Rong, T. Xu, J. Huang, W. Huang, H. Cheng, Y. Ma, Y. Wang, T. Derr, L. Wu, and T. Ma. Deep graph learning: Foundations, advances and applications. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 3555–3556, 2020.

[25] J. Su, D. Zou, Z. Zhang, and C. Wu. Towards robust graph incremental learning on evolving graphs. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 32728–32748. PMLR, 23–29 Jul 2023.

[26] L. Sun, J. Ye, H. Peng, F. Wang, and S. Y. Philip. Self-supervised continual graph learning in adaptive riemannian spaces. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 4633–4642, 2023.

[27] Z. Tian, D. Zhang, and H.-N. Dai. Continual learning on graphs: A survey. *arXiv preprint arXiv:2402.06330*, 2024.

[28] C. Wang, Y. Qiu, D. Gao, and S. Scherer. Lifelong graph learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 13719–13728, 2022.

[29] L. Wang, X. Zhang, H. Su, and J. Zhu. A comprehensive survey of continual learning: Theory, method and application. *arXiv preprint arXiv:2302.00487*, 2023.

[30] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger. Simplifying graph convolutional networks. In *International conference on machine learning*, pages 6861–6871. PMLR, 2019.

[31] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

[32] F. Xia, K. Sun, S. Yu, A. Aziz, L. Wan, S. Pan, and H. Liu. Graph learning: A survey. *IEEE Transactions on Artificial Intelligence*, 2(2): 109–127, 2021. doi: 10.1109/TAI.2021.3076021.

[33] S. Yan, J. Xie, and X. He. Der: Dynamically expandable representation for class incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3014–3023, 2021.

[34] P. Zhang, Y. Yan, C. Li, S. Wang, X. Xie, G. Song, and S. Kim. Continual learning on dynamic graphs via parameter isolation. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 601–611, 2023.

[35] P. Zhang, Y. Yan, C. Li, S. Wang, X. Xie, G. Song, and S. Kim. Continual learning on dynamic graphs via parameter isolation. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '23, page 601–611, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450394086. doi: 10.1145/3539618.3591652. URL https://doi.org/10.1145/3539618.3591652.

[36] X. Zhang, D. Song, and D. Tao. Cglb: Benchmark tasks for continual graph learning. *Advances in Neural Information Processing Systems*, 35:13006–13021, 2022.

[37] X. Zhang, D. Song, and D. Tao. Hierarchical prototype networks for continual graph representation learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(4):4622–4636, 2022.

[38] X. Zhang, D. Song, and D. Tao. Sparsified subgraph memory for continual graph representation learning. In *2022 IEEE International Conference on Data Mining (ICDM)*, pages 1335–1340. IEEE, 2022.

[39] X. Zhang, D. Song, and D. Tao. Ricci curvature-based graph sparsification for continual graph representation learning. *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[40] X. Zhang, D. Song, and D. Tao. Continual learning on graphs: Challenges, solutions, and opportunities. *arXiv preprint arXiv:2402.11565*, 2024.

[41] Z. Zhang, P. Cui, and W. Zhu. Deep learning on graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 34(1):249–270, 2020.

[42] X. Zheng, M. Zhang, C. Chen, Q. V. H. Nguyen, X. Zhu, and S. Pan. Structure-free graph condensation: From large-scale graphs to condensed graph-free data. *Advances in Neural Information Processing Systems*, 36, 2024.

[43] F. Zhou and C. Cao. Overcoming catastrophic forgetting in graph neural networks with experience replay. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 4714–4722, 2021.

[44] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun. Graph neural networks: A review of methods and applications. *AI open*, 1:57–81, 2020.