Continual Learning on Graphs: Challenges, Solutions, and Opportunities

Xikun Zhang, Dongjin Song*, *Member, IEEE*, and Dacheng Tao*, *Fellow, IEEE*

Abstract—Continual learning on graph data has recently garnered significant attention for its aim to resolve the catastrophic forgetting problem on existing tasks while adapting the existing model to newly emerged graph tasks. While there have been efforts to summarize progress on continual learning research over Euclidean data, such as images and texts, a systematic review of continual graph learning works is still absent. Graph data are far more complex in terms of data structures and application scenarios, making continual graph learning task settings, model designs, and applications extremely complicated. To address this gap, we provide a comprehensive review of existing continual graph learning works by elucidating the different task settings and categorizing the existing works based on their adopted techniques. We compare the continual graph learning works with traditional continual learning works and analyze the applicability of the traditional continual learning techniques to continual graph learning tasks. Additionally, we review the benchmark works that are crucial to continual graph learning research. Finally, we discuss the remaining challenges and propose several future directions.

Index Terms—Graph representation learning, continual learning, graph neural networks, continual graph learning, lifelong learning, lifelong graph learning, continual graph representation learning.

1 Introduction

Traditional graph learning typically assume the graph to be static. However, in most real world applications including both node level learning tasks and graph level learning tasks, the graphs will either grow in sizes or in quantities, and a desired model is expected to continually adapt to the newly emerging patterns without forgetting the knowledge previously learnt. For example, in a citation network, new categories of research papers (graph nodes) and the accompanying citations (graph edges) will constantly emerge, and a document classifier working on it is expected to be able to continually adapt to the distribution of new data while maintaining the learnt knowledge of the previously observed data distribution at the same time [1], [2]. For drug discovery research, new molecule properties and new molecule categories may be discovered intermittently, and the molecule property predictor has to fit its parameters for new patterns without losing its capability for prediction on the previous molecule categories or properties [1], [3]. In such continual learning scenarios, a naive approach is to only train the model with the new data whenever a new task comes. But a model continually adapting to new tasks would suffer from the catastrophic forgetting problem, which is the severe performance decrease on previous tasks after learning new tasks. Another intuitive approach is to retrain the model over the entire dataset containing all previously observed data. However, due to the high retraining cost and potential privacy issue, this strategy is infeasible. In light of this, Continual Graph Learning (CGL), which aims to continually learn new tasks without forgetting previously learnt knowledge, has recently received increasingly more attention. Due to the complexity of graph data, existing continual learning works are highly heterogeneous in terms of the targeted graph type, learning settings, basic techniques, and evaluation metrics. First, the technical works may propose general techniques or techniques specialized for certain application scenarios (*e.g.* knowledge graph, recommender system, etc.). Second, according to the availability of the task identity during testing, continual graph learning works may adopt different settings including task-incremental (task-IL), domain-incremental (domain-IL), and class-incremental (class-IL). Besides the incremental setting, the graph learning tasks may also focus on different granularity including node-level tasks and graph-level tasks. Third, the techniques adopted by different works follow different mainstreams including the regularization, memory replay, and parameter-isolation. Finally, different from standard learning settings, the performance of a continual graph learning model is concerned with different perspectives including the overall performance, the performance decrease (forgetting), the inter-task interfere, etc. Therefore, depending on the research target, different works may adopt different metrics for evaluating the models. These orthogonal dimensions of continual graph learning works are baffling and create great barrier for researchers when entering the field.

To this end, in this paper, we provide a systematic review on the existing methods from the above mentioned perspectives. The rest of the paper is organized as follows. In Section 3, we explain the problem setup of continual graph learning from the perspectives of basic concepts, backbone model framework, task sequence construction, task granularity, incremental scenarios, as well as the evaluation metrics. In Section 4, we first systematically review the technical works following the three mainstreams including the regularization based, memory-replay based, and parameter-isolation based

- *Mr X. Zhang and Prof D. Tao are with the School of Computer Science, in the Faculty of Engineering, at The University of Sydney, 6 Cleveland St, Darlington, NSW 2008, Australia. Email: xzha0505@uni.sydney.edu.au, dacheng.tao@sydney.edu.au.*
- *Dr. D. Song is with the Department of Computer Science and Engineering, University of Connecticut, Storrs, Connecticut, the United States. Email: dongjin.song@uconn.edu.*
  ** indicates corresponding authors.*

methods, and then analyze the applicability of traditional continual learning techniques. In Section 6, we introduce the recently proposed benchmark works which aim to provide a consistent experimental setting and fair performance comparison platform. Then, we analyze the applicability of the traditional continual learning techniques to continual graph learning tasks. Finally, we also cover the benchmark works which provide a fair platform for comparing different techniques. Finally, in Section 7, we discuss the existing challenges and promising future directions for CGL research.

2 Background

2.1 Continual Learning

Continual learning [3], [4], [5], [6] targets the catastrophic forgetting problem, *i.e.* the phenomenon that a model's performance on previous tasks decreases drastically after learning the subsequent new tasks. During training, the learning can be formulated as training a model consecutively on a sequence of tasks. When learning each task, the model can only access the data of the current task, while the access to previous tasks is not allowed. In continual learning, the data of different tasks are typically from different distributions, *e.g.* images of different classes. Accordingly, after learning a new task, since the model parameters are freshly adapted solely for the new task, the learnt knowledge of the previous tasks may be overwritten, leading to the catastrophic forgetting on previous tasks. During testing, the model will be tested on each learnt task. A desired model should perform well on every learnt task. The evaluation of continual learning models are detailed in Section 5.

2.2 Graph Representation Learning

Graph representation learning [7], [8], [9], [10], [11], [12], [13], [14] aims to generate qualified representations for nodes, edges, or the entire graph, which can further be used for downstream tasks like node classification [7], [15]. link prediction [16], [17], [18], graph classification [19], [20], *etc.*. Graph neural networks (GNNs) are currently the state-of-the-art approaches. To obtain either node/edge level or graph level representations, GNNs would first generate node/edge level representations. The most popular GNNs follow the message passing neural network (MPNN) framework [19], and can be formulated as,

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

$$\mathbf{h}_v^{l+1} = \mathrm{U}_l(\mathbf{h}_v^l, \mathbf{m}_v^{l+1}; \boldsymbol{\theta}_l^{\mathrm{U}}), \tag{2}$$

As shown in the formulations above, different from learning on independent data (*e.g.* images), generating representations on graph data requires properly capturing the valuable topological information (*e.g.* through the message passing). Similarly, compared to classic continual graph learning on independent data, continual graph learning also have to properly consider preserving the highly valuable topological information. Besides, the topological connections may cause some continual learning techniques to be inapplicable. For example, to generate the representation of a single node, the message passing based GNNs would require aggregating information from multi-hop neighbors, and the memory replay based methods that store individual data become inapplicable.

2.3 Differentiation from Other Related Works

It is crucial to distinguish between CGL, dynamic graph learning [21], [22], [23], [24], [25], [26], [27], [28], [29], and few-shot graph learning [30], [31], [32], [32], [33], [34]. Dynamic graph learning primarily aims to capture the evolving graph structure and maintain up-to-date graph representations, with access to all prior information, rather than addressing the forgetting issue. Conversely, CGL focuses on the forgetting problem, and the previous task data are typically unavailable. An exception is CGL with inter-task edges, which permits aggregation of past task information via inter-task edges during the GNNs' neighborhood aggregation process. Nevertheless, the labels of prior task data remain inaccessible. Few-shot graph learning is designed for rapid model adaptation to new tasks. In training, few-shot learning models can access all tasks simultaneously, which is not the case for CGL (CGL with inter-task edges deviates slightly). During evaluation, few-shot learning models are tested on new tasks after initial fine-tuning, whereas CGL models are evaluated on existing tasks without any fine-tuning.

3 Problem Setups

Different from continual learning on Euclidean data, Continual Graph Learning (CGL) is concerned with more complex task configurations. For example, in some applications, the learning is on a growing graph, and each new task is a new subgraph attaching to the existing graph. While in other scenarios, the task may be graph-level ones, and each new task is a set of independent individual graphs. In this section, we first provide a general formulation of the continual learning process, based on which we then derive different specific learning scenarios.

To generally cover different graph evolving scenarios, the contniual learning process can be formulated as on a sequence of graphs: $\mathcal{S} = \{\mathcal{G}_1, \mathcal{G}_2, ..., \mathcal{G}_T\}$. Each graph $\mathcal{G}_\tau$ represents the entire graph that has grown from task 1 to τ. Each $\mathcal{G}_\tau$ consists of a node set $\mathbb{V}_\tau$ containing all the nodes, and an edge set $\mathbb{E}_\tau$ containing all the edges. $\mathbb{E}_\tau$ can also be represented as an adjacency matrix $\mathbf{A}_\tau \in \mathbb{R}^{|\mathbb{V}_\tau|\times|\mathbb{V}_\tau|}$. Each entry $\mathbf{A}_\tau^{u,v}$ denotes the edge between node u and v ($\mathbf{A}_\tau^{u,v} = 0$ if node u and v are not connected by an edge). The number of edges connected to a node is referred to as its degree, and the degree of all nodes with in a graph can be stored as the diagonal entries of a degree matrix $\mathbf{D}_\tau \in \mathbb{R}^{|\mathbb{V}_\tau|\times|\mathbb{V}_\tau|}$, *i.e.* $\mathbf{D}_\tau^{u,u}$ is the degree of node u. In practice, $\mathbf{D}_\tau$ is often used to normalize the adjacency matrix, *i.e.* $\hat{\mathbf{A}}_\tau = \mathbf{D}_\tau^{-\frac{1}{2}}\mathbf{A}_\tau\mathbf{D}_\tau^{-\frac{1}{2}}$.

To conveniently refer to the nodes coming in different tasks, we denote the new nodes in a task τ as $\mathbb{V}_\tau^{new}$. Accordingly, we denote the induced subgraph based on $\mathbb{V}_\tau^{new}$, *i.e.* all nodes in $\mathbb{V}_\tau^{new}$ together with all edges that connect nodes in $\mathbb{V}_\tau^{new}$, as $\mathcal{G}_\tau^{new}$. $\mathcal{G}_\tau^{new}$ is the task-specific subgraph of the new task τ. Then, all the edges within each task specific subgraph are referred to as intra-task edges [3], while the other edges are inter-task edges connecting nodes across different tasks. We denote the intra-task edges of task τ as $\mathbb{E}_\tau^{intra}$, and the inter-task edges after the arrival of task τ as $\mathbb{E}_\tau^{inter} = \mathbb{E}_\tau \setminus \bigcup_{i=1}^{\tau} \mathbb{E}_\tau^{intra}$.

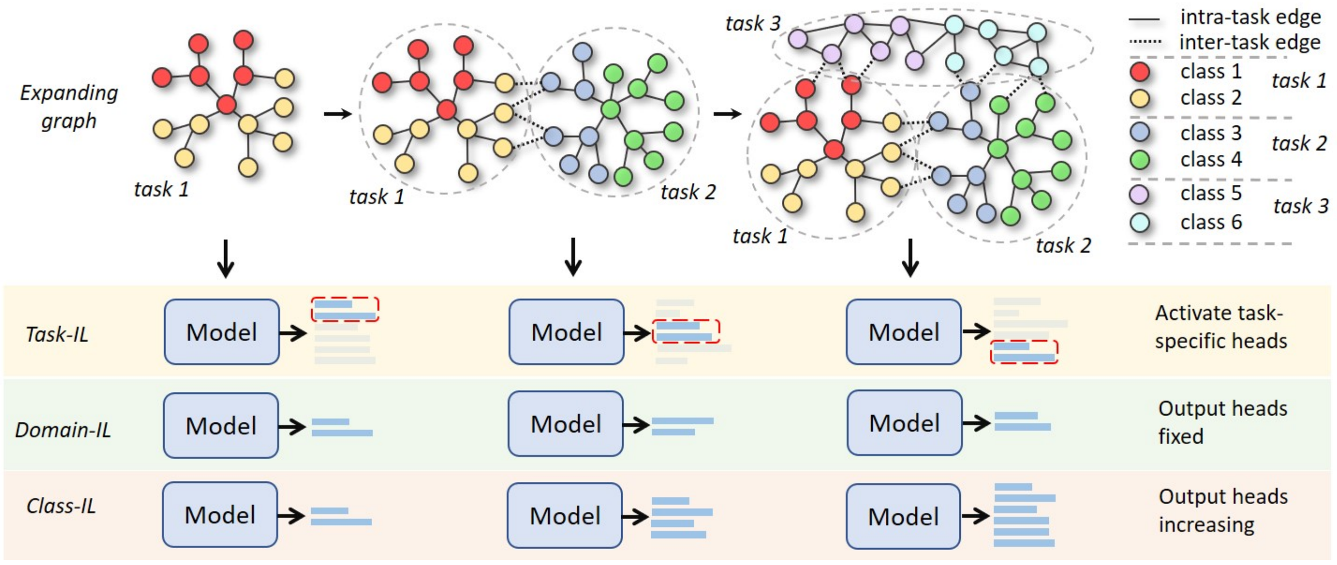

Fig. 1: Illustration of the different incremental settings.

3.1 Different Prediction Granularity

Different from learning on independent data, graph related tasks have different granularity, and correspond to different real-world scenarios and require different task sequence constructions.

Node/edge-level learning focus on generating representations and predictions for individual nodes/edges. For example, given a citation network, classifying the papers is modelled as a node classification task (node-level). With a protein interaction network, in which the nodes are proteins and the edges are interactions between proteins, predicting the missing interactions is modelled as a link prediction (edge-level) task. Since nodes and edges belong to the same granularity and edge related tasks are often based on the learnt node representation, we will use node-level to denote both node- and edge-level tasks in the following for simplicity.

Formally, depending whether inter-task edges are allowed to be preserved, node-level tasks can have two different formulations. First, without the inter-task edges, the representation of a task τ is:

$$\{\mathbf{h}_v | v \in \mathbb{V}_\tau^{new}\} = \mathrm{f}(\mathcal{G}_\tau^{new}; \boldsymbol{\theta}_\tau), \tag{3}$$

where $\mathbf{h}_v$ is the representation of node v generated by the model $\mathrm{f}(\cdot; \boldsymbol{\theta}_\tau)$ parameterized by $\boldsymbol{\theta}_\tau$ (the model parameters during learning task τ). When the inter-task edges are preserved, then the input for learning task τ is not limited to $\mathcal{G}_\tau^{new}$. Since GNNs could aggregate information from previously observed nodes via the inter-tasks, the input becomes the entire graph $\mathcal{G}_\tau$, *i.e.*,

$$\{h_v | v \in \mathbb{V}_\tau^{new}\} = \mathrm{f}(\mathcal{G}_\tau; \boldsymbol{\theta}_\tau). \tag{4}$$

When dealing with node-level tasks, the incoming subgraph $\mathcal{G}_\tau^{new}$ is typically a connected graph. While for graph-level tasks, $\mathcal{G}_\tau^{new}$ is a set of disconnected graphs, e.g. a set of molecule graphs. In this case, the graph-level representations can be formulated as:

$$\{\mathbf{h}_g | g \in \mathcal{G}_\tau^{new}\} = \mathrm{f}(\mathcal{G}_\tau^{new}; \boldsymbol{\theta}_\tau), \tag{5}$$

where g denotes the disconnected components in $\mathcal{G}_\tau^{new}$. Since graph-level continual graph learning deals with individual graphs (disconnected components), there is not inter-task edge across different tasks.

3.2 Different Incremental Scenarios

According to whether the task identities are provided during testing and whether a model is required to figure out the task identities, continual graph learning, as well as classic continual learning, can be categorized into task-incremental learning (task-IL), domain-incremental learning (domain-IL), and class-incremental learning (class-IL).

3.2.1 *Task-IL*

In task-IL, the task identities are revealed to the model during testing, therefore the model is not required to identify the given tasks. For classification tasks, existing models typically increase their output dimensions to accommodate new tasks, and only activate the corresponding dimensions for each given task during testing. For example, in molecular property prediction tasks, each new task could be predicting the existence of a new property.

3.2.2 *Domain-IL*

Domain-IL is more challenging than task-IL, since it does not provide task identities during testing. But it does not require to identify the given testing tasks either. In this scenario, the semantic meaning of a model's output dimensions are typically fixed and new tasks are viewed as data from new domains. For example, one possible scenario for continual learning on knowledge graphs is to sequentially learn on graphs with different entities and relations, while the prediction task is always the completion of the triplets.

Some works also consider time-incremental scenario, which splits time series data into different periods as different tasks. However, since the semantics of the model's output are same across different tasks, this scenario is essentially an instantiation of domain-IL, and different time periods are different domains in essence.

3.2.3 Class-IL

Class-IL is the most challenging among the three scenarios. During testing, task identities are inaccessible and the model has to identify the given tasks. For classification tasks, a model typically increases the output dimensions when new classes arrive, and have to pick out the correct class among all learnt classes, unlike task-IL that only requires distinguishing between classes within a known task.

4 Methods

Similar to traditional continual learning, continual graph learning also approach the problem from the perspectives of restricting the change in the model parameters, isolating and protecting the parameters that are important for previously learnt tasks, and replaying representative data from previous tasks to remind the model of the previously learnt patterns. However, a key challenge of continual graph learning is the necessity to properly preserve the topological structure of the data, which is crucial information contained in the graph data.

4.1 Regularization based methods

Since the reason of the forgetting is that the model parameters trained for previous tasks are modified after being adapted to new tasks, traditional regularization based methods [48], [49] add penalty terms to prevent the parameters from being drastically changed. However, these methods do not explicitly preserve the topology of the graph data. Targeting this insufficiency, topology-aware weight preserving (TWP) [1] proposes to explicitly preserve the topology learnt on previous tasks via regularization on the model weights.

Denoting the model parameters after learning the t-th task as $\boldsymbol{\theta}_t$, the learning of regularization based methods on the $(t+1)$-th can be generally formulated as,

$$\boldsymbol{\theta}_{t+1} = \arg\min_{\boldsymbol{\theta}} \mathcal{L}_{t+1} + \sum_{n=1}^{t} I_n \otimes (\boldsymbol{\theta} - \boldsymbol{\theta}_n^*)^2. \tag{6}$$

In Equation 6, $\boldsymbol{\theta}_n^*$ is the optimal parameters for the n-th task, and the importance scores I_n indicate the importance of each parameter for the performance of task n. In other words, the change of the parameters that are important to previous tasks are strongly penalized, while the less important parameters are more free to adapt to the new tasks. In different methods, I_n are calculated with different strategies. In TWP, the importance scores consist of two parts. The first part is calculated for each weight of the model based on the sensitivity of the loss on the weight, which is measured with the gradient of the loss with respect to the weight. This part resembles the strategies adopted by EWC [48] and MAS [49], and serves to preserve the weights that are important to previous tasks. We denote the corresponding importance scores as I_n^p. The second part, which is the key contribution of TWP, is the sensitivity of the learnt graph structure on the weight. For backbones like graph attention network (GAT) that explicitly learns the strength of the edges via the attention scores, the learnt structures are deemed as the attention score on each weight, and the sensitivity is measured as the gradient of the squared l_2 norm of the attention scores with respect to the weights. Denoting the attention score between two nodes i and j as e_{ij}, the attention score between a node i and its all neighbors can be represented as $\mathbf{e}_i = [e_{i1}, ..., e_{i|\mathcal{N}_i|}]$. After that, the sensitivity (importance) of the topology learnt in task n with respect to the model parameters is defined as the gradient of the squared $l2$ norm with respect to the parameters.

$$I_n^t = \frac{\partial(||[\mathbf{e}_1, ..., \mathbf{e}_{|V^n|}]||)}{\partial \boldsymbol{\theta}} \tag{7}$$

For backbones without attention mechanism, the attention score between two nodes is calculated with a non-parametric attention mechanism,

$$e_{ij} = \mathbf{h}_i^T \tanh(\mathbf{h}_j) \tag{8}$$

With the importance scores for both the model performance and the topology, the final importance is obtained by their weighted summation,

$$I_n = \lambda_p I_n^p + \lambda_t I_n^t. \tag{9}$$

Besides the learnt attention scores, the graph curvature is another important topological property of the graph data, and is crucial for graph learning [50], [51]. When learning on a sequence of graphs, considering that the curvature of the incoming graph may constantly change, Riemannian Graph Continual Learner (RieGrace) [35] further proposes Adaptive Riemannian GCN (AdaRGCN) for accommodating the change in the graph curvature and Label-free Lorentz Distillation for alleviating the forgetting problem. Specifically, AdaRGCN can constantly adapt and capture the topology of the given graphs with different curvatures. And the Label-free Lorentz Distillation serves to maintaining the learnt knowledge when learning on the graph sequence. Considering the possible scarcity of labels in real worlds applications, Label-free Lorentz Distillation trains a model based on contrastive learning without any label. It consists of two parts, including a cross-layer intra-distillation and a cross-model inter-distillation. The intra-distillation serves to learn the pattern of the current graph, and the contrastive learning is conducted by maximizing the agreement between the representations of each node at different layers of the model. Formally, the objective function is

$$\mathcal{J}(\mathbf{x}_i^{s,L}, \mathbf{x}_i^{s,H}) = \log \frac{\exp \mathrm{Sim}^{\mathcal{L}}(\mathbf{x}_i^{s,L}, \mathbf{x}_i^{s,H})}{\sum_{j=1}^{|\mathcal{V}|} \mathbb{I}\{i \neq j\} \exp \mathrm{Sim}(\mathbf{x}_i^{s,L}, \mathbf{x}_i^{s,H})}. \tag{10}$$

In Equation 10, $\mathbf{x}_i^{s,L}$ and $\mathbf{x}_i^{s,H}$ denote the low- and high-level representations of the i-th node, extracted by shallow and deep layers of the model. Since these two representations correspond to the same node, the objective is designed to maximize their similarity, which is measured by the adopted function $\mathrm{Sim}^{\mathcal{L}}(;)$. And $\mathcal{V}$ is the node set of the entire graph. Besides learning the pattern of the current graph, the knowledge learnt from the previous tasks should also be distilled into the current model, which is achieved by the inter-distillation loss,

$$\mathcal{J}(\mathbf{x}_i^{t,H}, \mathbf{x}_i^{s,H}) = \log \frac{\exp \mathrm{Sim}^{\mathcal{L}}(\mathbf{x}_i^{t,H}, \mathbf{x}_i^{s,H})}{\sum_{j=1}^{|\mathcal{V}|} \mathbb{I}\{i \neq j\} \exp \mathrm{Sim}(\mathbf{x}_i^{t,H}, \mathbf{x}_i^{s,H})}. \tag{11}$$

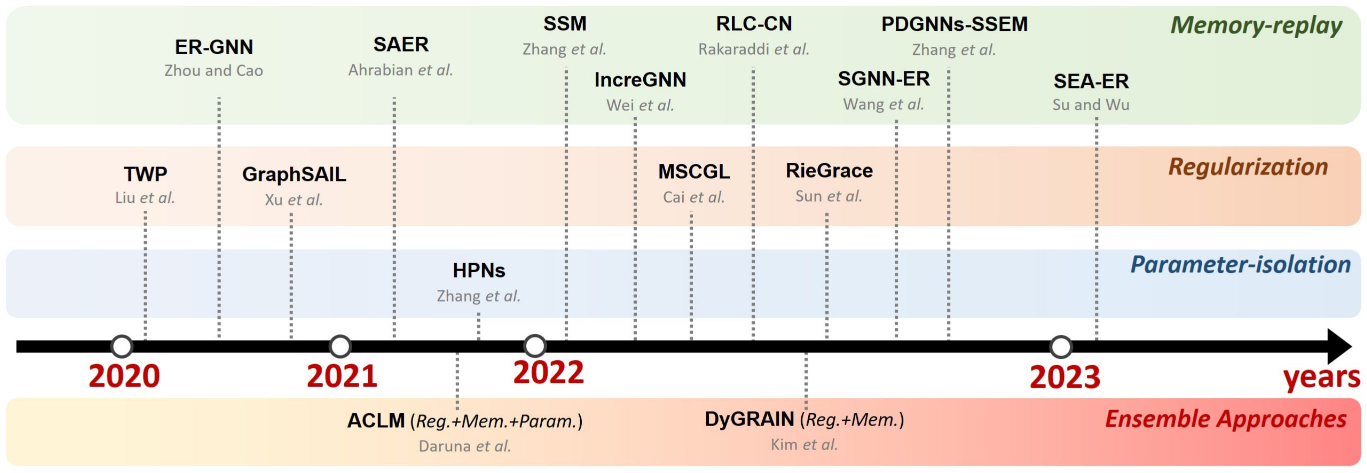

Fig. 2: Caption

Method	Applications	Task Granularity	Technique	Characteristics
TWP [1]	General	Node/Graph	Reg.	Preserve the topology learnt from previous tasks
RieGrace [35]	General	Node	Reg.	Maintain previous knowledge via knowledge distillation
GraphSAIL [36]	Recommender Systems	Node	Reg.	Local and global structure preservation, node information preservation
MSCGL [37]	General	Node	Reg.	Parameter changes orthogonal to previous parameters
ER-GNN [38]	General	Node	Mem.	Replay representative nodes
SSM [39]	General	Node	Mem.	Replay representative sparsified computation subgraphs
PDGNNs-SSEM [40]	General	Node	Mem.	Replay representative sufficient subgraph embeddings
IncreGNN [41]	General	Node	Mem.	Replay nodes according to their influence
RLC-CN [42]	General	Node	Mem.	Model structure adaption and dark experience replay
SGNN-ER [43]	General	Node	Mem.	Model retraining with generated fake historical data
SAER [44]	Recommender System	Node	Mem.	Buffer the representative user-item pairs based on reservoir sampling
SEA-ER [45]	General	Node	Mem.	Minimize the structural difference between the memory buffer and the original graph
HPNs [2]	General	Node	Para.	Extracting and storing basic features to encourage knowledge sharing across tasks, model expanding to accommodate new patterns
DyGRAIN [46]	General	Node	Mem.+Reg.	Alleviate catastrophic forgetting and concept shift of previous task nodes via memory replay and knowledge distillation
ACLM [47]	Knowledge Graph	Node	Mem.+Reg.+Para.	Adapting general CL techniques to CGL tasks

TABLE 1: Caption

Equation 11 has the same formulation as Equation 10, while the difference is that the contrastive learning is between the high-level representation of the teacher model and the student model. These two loss on each node are then summed up and balanced with a parameter λ, $\mathcal{J}_{overall} = \sum_{i=1}^{|\mathcal{V}|} \mathcal{J}(\mathbf{x}_i^{s,L}, \mathbf{x}_i^{s,H}) + \lambda \sum_{i=1}^{|\mathcal{V}|} \mathcal{J}(\mathbf{x}_i^{t,H}, \mathbf{x}_i^{s,H})$. By optimizing these two losses simultaneously, the current student model contains knowledge from both the current graph and the previously learnt model, and will serve as the teacher model when learning the next task.

The methods mainly focus on the forgetting problem. However, different from traditional continual learning, in which the new and old data are independent, the newly incoming data (graph nodes) in CGL could form edges connecting to the existing nodes. Since GNNs generate the prediction for a node based on its multi-hop neighbors, the newly formed connections between new and old nodes will cause concept drift for the old nodes and alter their representation [3]. Therefore, targeting both this challenge and the catastrophic forgetting problem, DyGRAIN [46] is designed to first identify the nodes that are most influenced by new nodes and the nodes that are most vulnerable to forgetting. After that, model retraining and knowledge distillation are applies to alleviate the influence of the new nodes and the forgetting problem. For the detection of the vulnerable nodes, the changing neighborhood problem is termed as the time-varying receptive field in the DyGRAIN paper. Based on the neighborhood aggregation of GNNs, the nodes most influenced by the changed receptive field are detected based on the adjacency matrix to conduct the neighborhood aggregation. Specifically, denoting the previously observed nodes as $\mathcal{M}$, the nodes in the t-th task as $\mathcal{V}^{(t)}$, an indicator matrix $\mathbf{V}$ can be derived to represented the influence of each node after certain rounds of message passing.

$$\mathbf{V}_{ii}^{(0)} = \begin{cases} 1 & if \quad i \in \mathcal{V}^{(t)} \\ 0 & if \quad i \in \mathcal{M}, \end{cases} \qquad \mathbf{V}^{(l)} = \mathbf{P}^{(t)}\mathbf{V}^{(l-1)}, \tag{12}$$

where $\mathbf{P}^{(t)} = \mathbf{A}^{(t)}\mathbf{D}^{-1}$ is the adjacency matrix $\mathbf{A}^{(t)}$ normalized by the degree matrix $\mathbf{D}$. By iteratively applying $\mathbf{P}^{(t)}$ for L times, the entries in $\mathbf{V}^{(L)}$ denote how much information (*i.e.* influence) would be propagated from each node to the L-hop neighbors. Accordingly, a Structural Influence (SI) score of a node i is defined as:

$$\pi_{SI}(i) = \sum_{j \in \mathcal{M} \cup \mathcal{V}^{(t)}} \sum_{l=1}^{L} \mathbf{V}_{ij}^{(l)}. \tag{13}$$

Based on the structural influence score defined above, a subset of k nodes with the highest scores (*i.e.* $\mathcal{S}_{struct}$) are selected as the ones that are most vulnerable to the influence of the new nodes. Besides the purely structural influence, the authors also consider the influence based on the change in the node embeddings. Denoting the GNN model as $\mathrm{f}_{\boldsymbol{\theta}}(\cdot)$, the representation of a node i after the arrival of task t can be denoted as, $\mathbf{h}_i^{(t)} = \mathrm{f}_{\boldsymbol{\theta}}(\mathbf{x}_i, \mathbf{A}^{(t)}; \boldsymbol{\theta}^{(t-1)})$. And the influence of the t-th task is derived based on the change in the node embeddings and defined as the Feature Influence (FI) score,

$$\pi_{FI}(v_i) = 1 - \frac{\mathbf{h}_i^{(t-1)} \cdot \mathbf{h}_i^{(t)}}{||\mathbf{h}_i^{(t-1)}|| \cdot ||\mathbf{h}_i^{(t)}||}. \tag{14}$$

Similarly, with the obtained feature influence score, the top k nodes, *i.e.* $\mathcal{S}_{feat}$, are also selected as the vulnerable ones. Above all, to mitigate the influence from the time-varying receptive field, the union of the two sets of nodes, *i.e.* $\mathcal{S}_{IP} = \mathcal{S}_{struct} \cup \mathcal{S}_{feat}$, will be included in the memory for retraining the model.

Similar to the procedure to mitigate the influence of time-varying receptive field, the catastrophic forgetting is also dealt with by first identifying the most vulnerable nodes. Since the nodes within the L-hop neighbors are already sampled, they will be excluded during the identification of the nodes that are vulnerable to catastrophic forgetting, and the remaining node is denoted as $\mathcal{V}_C$. With $\mathcal{V}_C$, the loss on the previous task is,

$$\mathcal{L}^{(t-1)} = l(\mathcal{V}_C, \tilde{A}^{(t-1)}; \mathrm{f}_{\boldsymbol{\theta}^{(t-1)}}). \tag{15}$$

Since lower loss values correspond to the more representative nodes (the model is well adapted for them), from $\mathcal{V}_C$, a subset of nodes is selected by lower q percentile according to the loss value, $\mathcal{V}' = \{i \in \mathcal{V}_C | \mathcal{L}_i^{(t-1)} \leq P_q(\mathcal{L}^{(t-1)})\}$. To find the most vulnerable nodes, the first step is to estimate the new loss in task t of the nodes $\mathcal{V}'$ as,

$$\hat{\mathcal{L}^{(t)}} \approx l(\mathcal{V}', \tilde{\mathbf{A}}^{t}; \mathrm{f}_{\boldsymbol{\theta}^{(t-1)}}). \tag{16}$$

Then the loss-based importance score of a node i is defined as the increase in the loss,

$$\pi_{CF}(i) = \hat{\mathcal{L}}_i^{(t)} - \mathcal{L}_i^{(t-1)}, \tag{17}$$

and the top k nodes are the ones that with the largest loss and are selected as the most vulnerable ones to the new nodes, *i.e.* $\mathcal{S}_{KD}$. The role of $\mathcal{S}_{KD}$ is two fold. First, knowledge distillation is conducted to protect the performance on these vulnerable nodes by minimizing the discrepancy between the node representations generated by the new and old models. Second, $\mathcal{S}_{KD}$ is also used for retraining the model together with the nodes in $\mathcal{S}_{IP}$. Above all, by retraining the model with the nodes in $\mathcal{S}_{IP}$, DyGRAIN is capable to ensure that the old nodes whose neighborhood (receptive field) is altered by new nodes with up-to-date representations. And through the knowledge distillation and model retraining with nodes in $\mathcal{S}_{KD}$, the catastrophic forgetting can also be alleviated.

Unlike the grid data, like the images, videos, and text, graph data has tremendous different branches with significantly different properties. Accordingly, besides the above works for general continual graph learning on any graph data, there are also methods specialized for a certain kind of graphs. For example, specially designed for incremental learning for recommender systems, Graph Structure Aware Incremental Learning (GraphSAIL) designs three knowledge distillation techniques to avoid the forgetting problem, including the local structure distillation, global structure distillation, and the self-embedding distillation. Since the nuclear operation of GCNs is the neighborhood aggregation over the local neighbors, preserving the local contextual structure is crucial. In GraphSAIL, this local structure is reflected through the affinity scores between the center nodes and their neighbors. And the affinity is maintained by knowledge distillation between the new and old tasks. Mathematically, it can be formulated as,

$$\mathcal{L}_{local} = \Big(\frac{1}{|\mathcal{U}| \sum_{u \in \mathcal{U}} (\mathbf{e}_u^{(t-1)} \cdot \mathbf{c}_{u,N_u^{t-1}} - \mathbf{e}_u^t \cdot \mathbf{c}_{u,N_u^{t-1}}^t)^2} \tag{18}$$

$$+ \frac{1}{|I|} \sum_{i \in \mathcal{I}} (\mathbf{e}_i^{t-1} \cdot \mathbf{c}_{i,N_i^{t-1}}^t)^2\Big), \tag{19}$$

where $\mathcal{U}$ and $\mathcal{I}$ are the user set and item set, $\mathbf{e}_u^t$ is the embedding of the user u during learning task t, and c denotes the average of the neighboring user nodes or item nodes,

$$\mathbf{c}_{u,N_u^{t-1}} = \frac{1}{|\mathcal{N}_u^{(t-1)}|} \sum_{i' \in N_u^{t-1}} \mathbf{e}_{i'}^t, \tag{20}$$

$$\mathbf{c}_{i,N_u^{t-1}} = \frac{1}{|\mathcal{N}_i^{(t-1)}|} \sum_{u' \in N_i^{t-1}} \mathbf{e}_{u'}^t. \tag{21}$$

Besides the local structure denoting the context of a node, the global position of a node within the graph is also crucial. For example, the distance among the users can reflect certain preference groups. Therefore, a global structure distillation is also designed. Specifically, to capture the global structure, a set of anchor nodes is first calculated for both the users and the items based on K-means clustering algorithm. Then each node has a similarity distribution over all the anchors denoting the relative position of the node within the graph,

$$GS_{u,\mathcal{A}_u^{t,k}}^t = \frac{e^{\mathrm{SIM}(\mathbf{e}_u^t, \mathcal{A}_u^{t,k})}/\tau}{\sum_{k'=1}^{K} e^{\mathrm{SIM}(\mathbf{e}_u^t, \mathcal{A}_u^{t,k'})/\tau}}, \mathrm{SIM}(\mathbf{a}, \mathbf{b}) = \mathbf{a}^{\mathrm{T}}\mathbf{b}, \tag{22}$$

where $\mathcal{A}_u^t$ denotes the anchors calculated during task t. And the goal of global structure preserving is to maintain the similarity distribution across different tasks,

$$\mathcal{S}_{u,\mathcal{A}_u} = D_{KL}(GS^t_{u,\mathcal{A}_u^t} || GS^{t-1}_{u,\mathcal{A}_u^{t-1}}). \tag{23}$$

Finally, besides the topological information including both the local and global structures, the information contained in each user/item node is also highly valuable. To protect this part of information, a self-distillation is proposed,

$$\mathcal{L}_{self} = (\frac{1}{|\mathcal{U}|}\sum_{u\in\mathcal{U}} \frac{\eta_u}{||\eta_U||_2}||\mathbf{e}_u^{t-1} - \mathbf{e}_u^t||_2 \tag{24}$$

$$+\frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}} \frac{\eta_i}{||\eta_I||_2}||\mathbf{e}_i^{t-1} - \mathbf{e}_u^t||_2), \tag{25}$$

where $\eta_u = \frac{|\mathcal{N}_u^{t-1}|}{|\mathcal{N}_u^t|}$ and $\eta_i = \frac{|\mathcal{N}_i^{t-1}|}{|\mathcal{N}_i^t|}$ are the normalizing factors.

Multi-modal Structure-evolving Continual Graph Learning (MSCGL) [37] is proposed to tackle the challenges of continual graph learning with multi-modal data. Specifically, MSCGL consists of two parts, including a Neural Architecture Search (NAS) module serving to adaptively optimize the model architecture for accommodating new tasks, and a Group Sparse Regularization (GRS) module for preserving crucial information of the previously learnt tasks. The search space of the model architecture is designed as multiple choices of different GNNs, in which the different modalities are processed with separate GNNs. Denoting the distribution of the model architecture a as $P(\mathrm{a};\boldsymbol{\theta})$ parameterized by $\boldsymbol{\theta}$, the objective can be formulated as a bi-level optimization including the maximization of the expected accuracy $\mathcal{E}[\mathcal{R}(\mathrm{a}(w*, G))]$ and minimization of the training loss $\mathcal{L}_{train}(\mathrm{a}(w, G))$,

$$\max \mathcal{E}[\mathcal{R}(a(w*, G))], s.t. w* = \arg\min_{w} \mathcal{L}_{train}(a(w, G)). \tag{26}$$

With the model obtained from NAS, a Group Sparse Regularization (GSR) is proposed to sparsify the network. During the sparsification, two restrictions proposed by [52] are adopted to ensure that the parameter changes are block-sparse and orthogonal to the previous parameters, so that the obtained new structure has less forgetting problem on the previous tasks.

Besides the works proposing new techniques that are specially suitable for general continual graph learning, there are also works that explore how to apply existing techniques to tackle specific continual graph learning problems. For example, in fake news detection tasks, to continually accommodating new data without retraining the model on all previous data , which is prohibitively expensive, [23] adopts the Elastic Weight Consolidation (EWC) [48] and Gradient Episodic Memory (GEM) [53], which successfully protect the performance on both the previous data and new data empirically. Both EWC and GEM are model-agnostic continual learning methods, and do not explicitly consider the topological information within the graph data.

4.2 Memory Replay based Methods

Memory replay based methods prevent forgetting by retraining the model with representative data from previous tasks. Traditional continual learning methods process individual data without interactions, which can be simply sampled and stored in a buffer. However, for learning on graphs, memory replay based methods would meet the memory explosion challenge [3]. When generating the representation of a node (a datum), Graph Neural Networks (GNNs) typically aggregate information from multi-hop neighbors. Therefore, to regenerate the representation of a single node, the information from an exponentially expanding neighborhood have to be stored. On dense graphs, the memory consumption would easily become intractable [3]. Due to this challenge, Experience Replay Graph Neural Network (ER-GNN) [38] directly ignore the graph topology and only store the attributes of one single node for regenerating the representation. Denoting the node buffer as $\mathcal{B}$, the loss function of ER-GNN when learning the n-th task can be formulated as

$$\mathcal{L}_n(\boldsymbol{\theta}, \mathcal{D}_i^{tr}, \mathcal{B}) = \beta\mathcal{L}_n(\boldsymbol{\theta}, \mathcal{D}_i^{tr}) + (1-\beta)\mathcal{L}_n(\boldsymbol{\theta}, \mathcal{B}), \tag{27}$$

where

$$\beta = \frac{|\mathcal{B}|}{|\mathcal{D}_i^{tr}| + |\mathcal{B}|}. \tag{28}$$

In other words, with the memory buffer, the total training loss is a weighted summation of the loss calculated with the data from the current task and the buffered data. And the contribution of these two parts are balanced based on their relative sizes. To properly populate the memory buffer, three different strategies are proposed in ER-GNN to select the representative nodes, including Mean of Feature (MF), Coverage Maximization (CM), and Influence Maximization (IM). The MF strategy is developed based on the intuition that the average feature vector of each class is representative, therefore the selected nodes should be the ones that are the closest to the average feature vector. The average feature vector (prototype) for a class l can be obtained in two ways,

$$\mathbf{c}_l = \frac{1}{|V_l|}\sum_{(v\in V)} \mathbf{x}_v, \mathbf{c}_l = \frac{1}{|V_l|}\sum_{(v\in V)} \mathbf{h}_v, \tag{29}$$

where V_l denotes the node set of the class l, $\mathbf{x}_v$ is the input attributes of the node v, and $\mathbf{h}_v$ is the representation of v generated by the GNN. The second strategy, CM, aims to maximize the diversity of the selected nodes. Specifically, for each node in each task, the number of nodes from different classes within a certain distance d is first counted as,

$$\mathcal{N}(v) = \{u|dist(v, u) < d, y_v \neq y_u\}. \tag{30}$$

In equation 30, y_v denotes the label of node v, $dist(\cdot,\cdot)$ measures the distance between two nodes, and d is a specified threshold. The lower $|\mathcal{N}(v)|$ a node v has, the more distant node v is from the nodes from different classes. Therefore, to maximize the coverage of the selected nodes in the representation space, CM is designed to select the several nodes with the lowest $|\mathcal{N}(v)|$. The third strategy, IF, aims to find the nodes with the most influence on the model parameters. Accordingly, a naive approach to probe the influence of a node is removing it and retraining the model to check the parameter change, which is very inefficient. Therefore, based on an alternative approach [54] to replace the node removal with loss upweighting, the authors propose to approximate the importance of a training node by its influence on the loss of testing nodes. For a training node v and a testing node u, the influence of node v during task n is derived as,

$$I(v,u) = -\nabla_{\boldsymbol{\theta}}\mathcal{L}_n(u)^T\mathbf{H}_{\boldsymbol{\theta}}^{-1}\nabla_{\boldsymbol{\theta}}\mathcal{L}_n(v), \tag{31}$$

where $\mathbf{H}_{\boldsymbol{\theta}}$ is the Hessian matrix of the loss function. Finally, the influence on all testing nodes are summed up as the approximation of the node v.

Although being easy to implement, the valuable topological information is not preserved in ER-GNN. To preserve the graph topology and maintain a tractable memory consumption at the same time, Sparsified Subgraph Memory (SSM) [39] is proposed to store sparsified computation subgraphs. Given a computation subgraph to store, SSM would sample a fixed number of nodes iteratively from the 1-hop to a specified K-hop neighbors. This hop-by-hop sampling manner can ensure the connectivity of the sparsified subgraph. After storing the sparsified subgraphs in the memory buffer $\mathcal{SSM}$, the total loss for learning on a task τ is derived as,

$$\begin{aligned}\mathcal{L} = & \underbrace{\sum_{u\in\mathbb{V}_\tau} l(\mathrm{f}(\mathcal{G}_u^{sub};\boldsymbol{\Theta}),\mathbf{y}_u)}_{\text{loss of the current task } \mathcal{L}_\tau} \\ & + \lambda \underbrace{\sum_{\bar{\mathcal{G}}_v^{sub}\in\mathcal{SSM}} l(\mathrm{f}(\bar{\mathcal{G}}_v^{sub};\boldsymbol{\Theta}),\mathbf{y}_v)}_{\text{auxiliary loss } \mathcal{L}_{aux}}.\end{aligned} \tag{32}$$

SSM is a model agnostic method that can be implemented with any GNN. Another recently proposed memory based continual graph learning approach, PDGNNs-SSEM [40], formulates a framework to store the complete topological information of each computation subgraph with a single embedding vector. Specifically, PDGNNs-SSEM consists of a general GNN framework named as Parameter Decoupled Graph Neural Networks (PDGNNs) and a memory buffer named as Sufficient Subgraph Embedding Memory (SSEM). As analyzed by [39], the key challenge in storing complete topological information of graph data is the memory explosion problem, *i.e.* the size of the computation subgraph grows exponentially with the number of hops. Therefore, PDGNNs is proposed to decouple the trainable parameters from the input computation subgraph, so that the nodes in the computation subgraph do not interact with the trainable parameters individually, and retraining the model only needs an overall information of each computation subgraph. Specifically, instead of iteratively aggregating the neighboring nodes and conducting node feature transformation, PDGNNs first encode the entire computation subgraph into a sufficient subgraph embedding with a non-parametric function to capture the topological information of the computation subgraph,

$$\mathbf{e}_v = \mathrm{f}_{topo}(\mathcal{G}_v^{sub}). \tag{33}$$

In Equation 33, $\mathbf{e}_v$ denotes the sufficient subgraph embedding (SSE) for node v, $\mathcal{G}_v^{sub}$ is the computation subgraph of node v, and $\mathrm{f}_{topo}(\cdot)$ is the non-parametric function that can be instantiated with different forms. In the paper, both linear formulation and non-linear formulation of $\mathrm{f}_{topo}(\cdot)$ are explored. After that, the obtained SSEs are then fed into a trainable function to generate the final output prediction,

$$\hat{\mathbf{y}}_v = \mathrm{f}_{out}(\mathbf{e}_v;\boldsymbol{\theta}). \tag{34}$$

With the PDGNNs framework, for retraining the model with a certain node v, we no longer need the entire computation subgraph $\mathcal{G}_v^{sub}$. Instead, we can only store the SSE of v for re-training the trainable function $\mathrm{f}_{out}(\cdot)$. Accordingly, the Sufficient Subgraph Embedding Memory ($\mathcal{SSEM}$) is developed to store the selected SSEs for each task,

$$\mathcal{SSEM} = \mathcal{SSEM} \bigcup \mathrm{sampler}(\{\mathbf{e}_v \mid v \in \mathbb{V}_\tau\}, n), \tag{35}$$

where $\mathrm{sampler}(\cdot,\cdot)$ is the chosen sampling strategy for populating the memory buffer, n is the budget for storage, and $\mathbb{V}_\tau\}$ denotes the node set of the current task. Compared to ER-GNN and SSM, SSEM can maintain complete topological information for each node with only one emebdding vector, therefore is highly efficient in terms of both space complexity and computation complexity during memory replay. But SSEM has to be accompanied with a GNN following the PDGNNs framework, while ER-GNN and SSM can be implemented with any GNN.

As mentioned before in Section 4.1, the inter-task edge connections would alter the neighborhood of the nodes in the previous tasks and cause concept shift, while the methods mentioned above do not explicitly consider this problem. Targeting this challenge, Structure-Evolution-Aware Experience Replay (SEA-ER) [45] is proposed to explicitly consider the evolution of the graph structure when populating the memory buffer. Denoting the memory buffer for the j-th task as P_j, the objective for populating the buffer is to maximize the structural similarity between the selected samples and the rest nodes,

$$\min_{P_j\subset\mathcal{V}_j}\max_{u\in\mathcal{V}_j\setminus P_j}\min_{v\in P_j} d_{\mathrm{spd}(u,v)}, s.t.|P_j| = b, \tag{36}$$

where $d_{\mathrm{spd}(u,v)}$ denotes the shortest path distance between node u and v, b is the memory budget for task j, and $\mathcal{V}_j$ is the node set of the task j. Since the evolution of the graph structure may also influence the distribution of the data in previous tasks, besides the sampling strategy, an importance reweighting technique is also designed in SEA-ER to rescale the contribution of different nodes. Specifically, the objective function with the node reweighting for learning on task i is formulated as,

Similar to SEA-ER, Incremental Graph Neural Networks (IncreGNN) [41] also considers the influence of inter-task edges on nodes in the previous nodes. To tackle this challenge, IncreGNN not only proposes an approach to evaluate the node importance based on both their to the new nodes and their influence in the original graph, but also adopts regularization technique to further boost the performance. Specifically, the newly incoming graph data is denoted as the base change group. Based on the base change group, the 1-hop neighbors of the base change group in the previous task graph is defined as the first order change group, the 1-hop neighbors of the first order change group is defined as the second order change group, and so forth. Since the influence of the new nodes on the previous nodes decreases with the increase of the order of the changing group, the ratio of the budget for node sampling at different order of change group is designed as,

$$\frac{1}{i}/(1+\frac{1}{2},...,+\frac{1}{K}), \tag{37}$$

where i is the number of order that is being sampled, K is the total number of change groups. After defining the importance of the groups of nodes with different distance to the new nodes, the importance of an individual node v is evaluated by the personalized PageRank algorithm run on the previous task graph G^{t-1}, and is denoted as π_v. Then, denoting the number of nodes to sample at k-order of change group as n_k, the nodes at each order with higher personalized PageRank values are selected as the ones with more correlation with the new nodes,

$$I(G^t) = \cup_{k=1}^{K}\{v_i|\pi_{v_i} > \pi_{v_j}, i \in [1, n_k], j \notin [1, n_k]\}. \tag{38}$$

The above process only select the old nodes affected by the new nodes. To also maintain the knowledge learnt from the old nodes without inter-task edges to new nodes (unaffected nodes), the next step is to evaluate the importance of each unaffected node for sampling and storage. Specifically, K-means clustering is first conducted to divide the unaffected nodes into K clusters. Then, the node importance within each cluster is determined by the node degree, and the nodes with the highest degrees are selected and stored into the memory buffer. Finally, IncreGNN also adopts the Memory Aware Synapse (MAS) [49] as the regularization on the model parameters to further enhance the capability to retain learnt knowledge.

Different from the approaches above, which retrain the same model structure with buffered data, [42] proposes to also dynamically adjust the model to accommodate new tasks. The proposed model consists of two parts including a Reinforcement Learning based Controller (RLC) that decides the addition and pruning of the model structure, and a Child Network (CN) which is a evolvable GNN backbone controlled by the RLC. The RLC is based on reinforcement learning, which aims to learn an optimal policy $\pi(a_t|s_t))$ to choose the most appropriate action a_t to modify the model based on the model state at the t-the task, so that the expected summation of the future rewards $R_t = \sum_{k=0}^{\infty}\gamma_{t+k+1}^{k}$ is maximized. The decaying factor $\in (0, 1]$ serves to gradually shrink the contribution of the expected rewards at distant time steps. Accordingly, the Q-function of RLC for taking an action a_t with a model state s_t is,

$$Q_{\pi}(s_t, a_t) = \mathrm{E}_{\pi}[R_t|S = s_t, A = a_t]. \tag{39}$$

The RLC in [42] is designed as a LSTM, which outputs a set of m actions $a_{1:m}$ for adjusting the number of features in each of the m layers of the CN. The search space for the action is defined as the addition and deletion operation of the CN hidden layers. When RLC is properly trained, the optimal actions for adjusting the CN is formulated as,

$$\hat{a}_{1:m} =_{a\in\mathcal{S}} \mathbb{E}[R(a_{1:m}, s_t)]. \tag{40}$$

Besides the RLC to optimize the structure of CN, to further enhance the continual learning capability of the model, the Dark Experience Replay [55] is also also adopted after being extended to graph data. Specifically, besides storing the data and the labels of the selected examples, their logits output by the CN is also buffered. During replay, the model retraining loss would accordingly consists of two parts including the classification loss calculated with the buffered labels and data, as well as the knowledge distillation loss calculated based on the stored logits. The total loss including the loss for learning the new task is then formulated as,

$$\mathcal{L} = \mathcal{L}_t + \alpha||f_t(\mathcal{X}^i) - l_t||_2 + \beta\mathcal{L}_{cls}(f_t(\mathcal{X}^i, \mathcal{Y}^i)), \tag{41}$$

where $f_t(\cdot)$ denotes the current model, l_i is the logits stored from the i-the task, and $\mathcal{X}^i$ and $\mathcal{Y}^i$ are the data and labels stored from task i.

Besides the models that store real data from learnt tasks, in Streaming Graph Neural Networks via Generative Replay (SGNN-ER), an auxiliary generative model is adopted to retrain the GNN with generated fake historical data. Overall, the generative model would generates the neighborhood (in the form of a random walk with restart sequence) based on the GAN framework. Denoting the generator and discriminator parameterized by ϕ^t and φ^t at task t as G_{ϕ^t} and D_{φ^t}, the learning objective can be formulated as a confrontation between the generator and the discriminator,

$$\begin{aligned}\min_{\phi^t}\max_{\varphi^t} V(G_{\phi^t}, D_{\varphi^t}) = E_{v\sim p_{data}(v)}[\log D_{\phi^t}(v)] \\ + E_{z\sim p_z(z)}[\log(1 - D_{\varphi^t}(G_{\phi^t}(z)))].\end{aligned} \tag{42}$$

In SGNN-ER, the generated data is used for knowledge distillation between the old and new models to avoid forgetting, and the total loss can be formulated as

$$\begin{aligned}\mathcal{L} = r\mathrm{E}_{v\sim\mathcal{G}_A^t}[l(F_{\boldsymbol{\theta}^t}(v)), y_v] \\ + (1-r)\mathrm{E}_{v'\sim G_{\phi}^{t-1}}[l(F_{\boldsymbol{\theta}^t}(v'), F_{\boldsymbol{\theta}^{t-1}}(v'))],\end{aligned} \tag{43}$$

where $\mathcal{G}_A^t$ is the affected part at task t, including the newly added subgraph and the previous nodes connected to the newly added subgraph.

Originally, the GAN framework is for generating independent data. However, to generate the representation of a single node, GNNs would also take its multi-hop neighborhood with complex topological dependencies as input. Therefore, SGNN-ER proposes to generate random walk with restart (RWR) sequences, which is then converted into neighborhood subgraphs based on the generated edge connectivity. Considering the neighborhood of some previous nodes are changed by the newly added nodes, to avoid enforcing the patterns that already disappear because of the neighborhood change, SGNN-ER proposes a forgetting mechanism to filter out the nodes that are significantly affected by the new nodes during memory replay. The mechanism consists of two steps. First, a set of significantly affected nodes $\mathcal{V}_C^t$ is first detected based on the change in their representations when the graph grows from $\mathcal{G}^{t-1}$ to $\mathcal{G}^t$,

$$\mathcal{V}_C^t = \{v|||F_{\boldsymbol{\theta}^{t-1}}(v, \mathcal{G}^t) - F_{\boldsymbol{\theta}^{t-1}}(v, \mathcal{G}^{t-1})|| > \delta\}, \tag{44}$$

where δ is the threshold on the change of representations. Since $\mathcal{V}_C^t$ denote the severely affected nodes, among the generated fake nodes for replay, the ones with high similarity to the nodes in $\mathcal{V}_C^t$ are rejected with high probability, which is formulated as,

$$p_{reject}(v) = \max(p_{sim}(v, u), u \in \mathcal{V}_C^t) \times p_r, \tag{45}$$

where p_r is a hyperparameter controlling the total number of nodes to delete, and $p_{sim}(\cdot,\cdot)$ is the similarity function for measuring the similarity between two nodes,

$$p_{sim}(v,u) = \sigma(-||F_{\boldsymbol{\theta}^{t-1}}(v,\mathcal{G}^{t-1}) - F_{\boldsymbol{\theta}}(u,\mathcal{G}^{t-1})||). \quad (46)$$

The methods mentioned above are general continual graph learning methods that are applicable to any graph data. For certain graph related tasks, due to the special properties of the graph data, some specialized techniques may be developed. For example, specially designed for incremental learning in graph based recommender system, Structure Aware Experience Replay (SAER) [44] samples and stores representative user-item pairs based on reservoir sampling, so that the previously learnt user behaviour patterns can be maintained when learning new tasks. Specifically, the reservoir is a set of previously observed interactions denoted as $\mathcal{H}$. When a new set of interactions, *i.e.* $\mathcal{H}'$, is given, a subset $\hat{\mathcal{H}}$ will firstly be sampled from $\mathcal{H}$ according to an adopted sampling strategy. Then the model will be trained with $\mathcal{H}' \cup \hat{\mathcal{H}}$. In [44], two strategies are adopted. The first one is the uniform sampling,

$$\hat{\mathcal{H}} = \{(u,i) \sim U(\mathcal{H})\} s.t|\hat{\mathcal{H}}| = \gamma|\mathcal{H}|, \quad (47)$$

where γ is a hyperparameter controlling the size of the sampled set. The uniform sampling serves as a naive baseline, and a more complex strategy is also adopted to balance the sampled interactions according to their degree, the probability for sampling an interaction (u,i) is,

$$P(u,i) = \frac{1/d_u^T}{\sum_{\hat{u},\hat{i}\ \mathcal{H}} 1/d_{\hat{u}}^T}, \quad (48)$$

in which d_u is the degree of the user u, and the temperature T is a hyperparameter for regulating the smoothness of the probability distribution. The pairwise data structure is commonly adopted by recommender systems, but is not available for general graph data. Therefore this technique is not generally applicable.

Currently, memory replay based methods are among the most effective approaches and are easy to implement. But it requires extra space to store representative data.

4.3 Parameter isolation based methods

The final category, parameter-isolation based methods, protects the model's performance on previous tasks by entirely or partially separate the model parameters for different tasks. Existing parameter isolation based continual graph learning approaches are scarce, and a representative method is Hierarchical Prototype Networks (HPNs) that propose to dynamically increment feature extractors and prototypes for accommodating new patterns. Specifically, HPNs consist of a set of Atomic Feature Extractors (AFEs) for extracting basic features from the given data, and three levels of hierarchical prototypes for storing learnt patterns. Given an input node, the AFEs will first extract basic features based on both the node attributes and its neighborhood relationship. Accordingly, the AFEs consist of two parts. One set of AFEs, denoted as AFE_{node}, serve to generating atomic node embeddings based on node attributes. The other set, denoted as $\text{AFE}_{\text{struct}}$, serve to capture the topological structure based on the neighborhood of the given node, and generate atomic structure embeddings accordingly. The obtained atomic embeddings correspond to the most basic features, and will then be matched to the existing atom-level prototypes (A-prototypes) according to their cosine similarity.

$$\text{Sim}_{E\to A}(v) = \{\frac{\mathbf{e}_i^T\mathbf{p}}{||\mathbf{e}_i||_2||\mathbf{p}||_2}|\mathbf{e}_i \in \mathbb{E}_A^{\text{select}}(v), \mathbf{p} \in \mathbb{P}_A\}, \quad (49)$$

where $\mathbb{E}_A^{\text{select}}(v)$ contains the atomic embeddings of the node v, and $\mathbb{P}_A$ is the set of A-prototypes. When the cosine distance between an A-prototype and an embedding is smaller than a threshold t_A, the A-prototype and the embedding is regarded as matched. If an embedding cannot be matched to any existing prototype, then it is regarded as new knowledge and will be used to initialize a new prototype. After A-prototype matching, the matched A-prototype will be further embedded and matched to higher level prototypes including the node-level prototypes (N-prototypes) and the class-level prototypes (C-prototypes). Each node will be matched to one N-prototype denoting the overall information of the entire node, and one C-prototype representing the properties of the class it belongs to. The union of the selected prototypes at different level are the final representation of the node v. Besides a novel model, The authors of [2] also propose a theoretical framework to analyze the continual learning capability of a model and justify that the proposed HPNs can theoretically eliminate the forgetting problem when the hyperparameters are properly set.

4.4 Application of Traditional Continual Learning Techniques

Although not specialized for graph data, some of the traditional CL techniques for Euclidean data are both data- and model-agnostic, therefore are applicable to CGL. For example, regularization based methods like Elastic Weight Consolidation (EWC) [48] and Memory Aware Synapses (MAS) [49] that only add regularization terms to the model weights, memory replay based models like Gradient Episodic Memory (GEM) [53] that stores the gradients of the losses on previous tasks and clip the current gradients to avoid increasing the previous task losses, and parameter-isolation based methods like Progressive Neural Networks [56] and Supermasks in Superposition (SupSup) [57] that entirely or partially separate parameters for different tasks. The key characteristic of these works that makes them applicable to CGL is that they are agnostic of the model structure and the data structure. While the other methods not satisfying this criteria are not directly applicable to CGL. For example, most memory replay based methods would directly store representative data from previous tasks. However, on a growing graph, since GNNs generates the representation of one datum (node) not only based on itself but also on its multi-hop neighbors, and GNNs with different designs require information from neighborhood with different range, it is unclear which part of the graph should be stored.

Targeting the inapplicability of some of the traditional CL methods, Feature Graph Network (FCN) [58] is proposed to accommodate the data by transforming each node into an independent feature graph, so that the data structure is no longer an obstacle for applying traditional continual learning techniques. When generating the feature graph, the topological information is also implicitly encoded. FCN provides a general approach to enable traditional CL techniques in any graph data. But for some special graphs, like knowledge graph, since the data storage format is more tidy, traditional CL techniques are applicable with little modification. For example, considering that the knowledge graph in robotics applications need to be frequently updated with new information, [47] extends several continual learning techniques including Progress Neural Networks [56], Copy weight Re-init Deep Generative Replay [59] to knowledge graph embedding for avoiding learning all the concepts afresh. We denote the ensemble of methods proposed by [47] as Adapted Continual Learning Methods (ACLM) for convenience.

5 Evaluation of CGL Models

Different from standard learning setting that is only concerned with one task, the evaluation of continual learning models have to consider the model's performance on all learnt tasks. Therefore, the most through approach to evaluate a continual learning model is to show its performance on each previous task after learning each new task. Formally, this result could be represented as a performance matrix $\mathrm{M}^p \in \mathbb{R}^{T\times T}$ [3]. Each entry $\mathrm{M}^p_{i,j}$ is the model's performance on the j-th task after the model has learnt the first i tasks. The performance matrix contains the raw performance during the entire learning process. Based on the performance matrix, multiple different metrics are adopted by the researchers. For example, considering that the performance matrix is inconvenient for performance comparisons across different methods, the average performance (AP) and average forgetting (AF) can be used [3]. These two metrics are widely adopted in CGL as well as traditional continual learning works, while the names may be different in different works. For example, they are named as Average Accuracy (ACC) and Backward Transfer (BWT) in [53], [60], Average Performance (AP) and Average Forgetting (AF) in [3], [38] and Performance Mean (PM) and Forgetting Mean (FM) in [1]. Broadly speaking, the entries of the performance matrix may not only be accuracy and can also be other metrics like F1 scores. Therefore, we follow the names (AP and AF) in CGLB [3]. Given a task sequence of length T, AP is calculated as the average of the performance on all previous tasks after learning the entire task sequence $\frac{\sum_{j=1}^{T}\mathrm{M}^p_{T,j}}{i}$. The AF is calculated as the average of the forgetting (performance decrease). The forgetting has two slightly different versions in different works. First, the forgetting of a task j after learning the final task T could be defined as $f_j = \mathrm{M}^p_{T,j} - \mathrm{M}^p_{j,j}$, which indicates how much the performance decreases from the time when the model has just learnt task j to the time when the model has learnt all tasks [3], [53]. Another definition compares the final performance with the best performance ever obtained when the model is trained from task 1 to task T, *i.e.* $f_j = \max_{i=1,...,T-1}\mathrm{M}^p_{i,j} - \mathrm{M}^p_{T,j}$. Different from the first definition, this definition compare the final performance on task j with the best performance ever obtained on task j before learning on the final task T. Another difference between these two definitions is that the sign of the computation. With the definition of the forgetting, the average forgetting (AF) can be naturally defined as $\frac{\sum_{j=1}^{T-1} f_j}{T-1}$. Further, to reflect the learning dynamics when learning sequentially on the tasks, the AP and AF after learning each task can be calculated, *i.e.* $\left\{\frac{\sum_{j=1}^{i}\mathrm{M}^p_{i,j}}{i}|i=1,...,T\right\}$ and $\left\{\frac{\sum_{j=1}^{i-1}\mathrm{M}^p_{i,j}-\mathrm{M}^p_{j,j}}{i-1}|i=2,...,T\right\}$. Besides AP and AF, researchers are also interested in the forward transfer during learning on the task sequence, and a forward transfer is also defined [53] as, $\frac{\sum_{j=2}^{T}\mathrm{M}^p_{j-1,j}-\bar{b_j}}{T-1}$, where $\bar{b_j}$ denotes the performance on task j with random model initialization. Accordingly, $\mathrm{M}^p_{j-1,j} - \bar{b_j}$ reflects how does the learning from task 1 to task $j-1$ benefit the performance on task j, *u.e.* forward transfer.

6 CGL Benchmarks & Datasets

Since CGL is a newly emerging area, different works adopt different datasets and conduct experiments under different settings, the fair comparison across different methods becomes difficult, which greatly impede the development of the area. Accordingly, several benchmark works are proposed to provide a consistent setting and dataset protocol for developing and comparing CGL techniques. In 2021, an introductory benchmark for CGL with graph-level tasks is introduced [61]. The benchmark is constructed based on three datasets: MNIST, CIFAR10, and OGBG-PPA. MNIST and CIFAR10 are image datasets and originally for computer vision research. However, following the process in [62], the images are converted to graphs. Given an image, with the SLIC algorithm [63], the pixels within a small local area with homogeneous intensity are defined as a super-pixel, corresponding to a node in the converted graph. And the topology (graph edges) are constructed as the k-nearest neighbor adjacency matrix. For the task construction, both MNIST and CIFAR10 contains 10 classes in total, and the classes are divided into five 2-class tasks. The OGBG-PPA [64] dataset contains a set of protein association neighborhoods (subgraphs) extracted from the large protein protein interaction (PPI) network. OGBG-PPA consists of 37 classes, each of which corresponds to a taxonomy group. The 37 groups are also splitted into 5 tasks. The first task contains 17 classes while the other four classes contain 5 classes. Constructing the tasks by splitting the classes of the data is a common approach for CGL research, because the data in different classes generally follow different distributions, which easily cause catastrophic forgetting and is ideal for CGL tasks. Besides the benchmark task construction, experiments of 4 baselines including Naive, EWC, Replay, and LwF are conducted under the class-IL scenario. The accompanied code of this benchmark is provided through https://github.com/diningphil/continual_learning_for_graphs.

In 2022, the Continual Graph Learning Benchmark (CGLB) is proposed, which is a more comprehensive benchmark covering 4 node-level datasets and 3 graph-level datasets under both task-IL and class-IL scenarios. The node-level datasets include CoraFull-CL, Arxiv-CL, Reddit-CL, and Products-CL. Both CoraFull-CL and Arxiv-CL are constructed based on citation networks provided in the public datasets CoraFull [65] and OGB-Arxiv [64]. Reddit-CL is constructed from social network consisting of Reddit posts [66]. Products-CL is constructed from the OGB-Products dataset [64], which is based on the Amazon product co-purchasing network. In CGLB, to maximize the number of tasks and increase the continual learning difficulty, in CGLB, all of these node-level tasks are split into 2-class tasks. For node-level CGL tasks, the growth of the graph would bring in inter-task edges connecting the nodes in previous and new tasks, and cause concept drift to the previous nodes connected by the inter-task edges. Accordingly, whether to preserve the inter-task edges when increasing the tasks are studied separately in CGLB. All of these node-level benchmark datasets support both task-IL and class-IL learning scenarios. On the node-level benchmark datasets, preliminary experimental results of multiple baseline methods are also provided under different settings. For graph-level CGL tasks, three benchmark datasets are constructed in CGLB including SIDER-tIL, Tox21-tIL, and Aromaticity-CL. SIDER-tIL is constructed from the SIDER dataset [67], which contains 1,427 drugs falling into 27 different classes. Since SIDER is a multi-label dataset, it is naturally constructed into a task-IL dataset with 27 tasks. Tox21-tIL is constructed from the Tox21 dataset [67], which is a multi-label dataset containing 8,014 molecules with 12 labels. Similar to SIDER, Tox21 is also constructed into a task-IL dataset with 12 tasks. Aromaticity-CL is constructed from the PubChem BioAssay Dataset [68], which contains 3,945 molecules falling into 40 classes. Since PubChem BioAssay Dataset is a multi-class dataset, the constructed Aromaticity-CL contains 15 2-class tasks, and can be used for both task-IL and class-IL. In Aromaticity-CL, some classes with very few data are removed.

7 Opportunities, Future Directions and Discussions

As a newly emerging and fast growing area, research on CGL have made significant progress with performance improvement and advancing into more challenging and practical learning scenarios. However, there are still multiple challenges to tackle.

7.1 Trade-off between Effectiveness and Space Complexity

Among the three strategies, regularization based methods is the most memory efficient since it requires little extra storage space, while memory-replay based methods and parameter-isolation based methods require significant extra space for storing the representative data or expanded network structure. However, as shown in recent works including benchmark work [3] and technical works [2], [39], [40], [69], the regularization based method is not as effective as the other two approaches. The reason is that although the regularization term can effectively restrict the change of the model parameters for preventing the forgetting on previous, it also reduce the model's capacity for adapting to the new tasks. On the contrary, neither memory-replay based methods [39], [70], [71] nor the parameter-isolation based methods impose restriction onto the model parameters. Moreover, the parameter-isolation based methods can even expand the model and increase the capacity so that the model can better adapt to the new tasks [56], [57], [72]. For the memory-replay based methods, the larger the buffer, the more information can be preserved from the previous tasks and the better the performance. For the parameter-isolation based methods, more budget for model expansion is also preferred for better performance. In other words, currently, the CGL techniques have to maintain a trade-off between the space complexity and the effectiveness in preventing forgetting. When the number of tasks becomes large, how to achieve better performance with limited memory space budget is still challenging.

7.2 Dependency on the Task Boundaries

Existing CGL mostly assume the task boundaries are available during training. For regularization based methods, the task boundary is used for matching the stored parameter importance to the corresponding tasks. For memory replay based methods, task boundaries help allocate memory budget for different tasks. For parameter-isolation based methods, the boundaries are also crucial to determine when to expand the network. However, in real-world applications, data from different tasks may come in as a mixture without clear task boundary, causing multiple challenges. First, without the given task identity, how to properly detect the distribution shift or the emergence of new patterns so that the model or memory buffer can expand is still to be explored. Second, since the data from a certain task may not only emerge once, even if the distribution shift is detected, the incoming 'new' patterns may have already been learnt previously. In this case, if the model always treat the incoming data with distribution shift as a new task, much memory space will be wasted for maintaining duplicated knowledge. Therefore, how to properly manage the learnt knowledge so that the learnt knowledge is preserved both effectively and efficiently is a challenge to tackle.

7.3 Concept Drift of Nodes in Existing Tasks

As pointed in CGLB [3] and multiple technical works [41], [45], since the inter-task edges connecting nodes from old and new tasks appear after learning on the old tasks, the affected nodes in the old tasks would experience concept drift. Currently, the only solution in the existing works [41], [45] is to retraining the model on these affected nodes within the updated large graph, so that the nodes after concept drift are learnt by the model. This approach assume that all previous nodes including the affected nodes and their neighboring nodes are accessible during learning the new tasks. However, due to reasons like the privacy issue, nodes from the previous tasks may not always be accessible, which is why continual learning works typically forbid the access to the previously learnt data. In light of this, how to tackle the concept drift on the affected previous nodes without accessing previous nodes is a crucial problem, especially dense networks with large amount of newly emerging inter-task edges.

7.4 Task-wise Knowledge Transfer

Currently, the focus of the CGL research is still on alleviating the forgetting problem. However, as pointed out by [2], [53], besides avoiding the negative transfer (forgetting), we also expect the task-wise transfer to be positive. In other words, the knowledge learnt from one task is expected to

TABLE 2: Commonly used datasets for CGL.

Dataset	Related works	Link
CoraFull	TWP [1],SSM [39],PDGNNs-SSEM [40],RLC-CN [42],CGLB [3]	github.com/shchur/gnn-benchmark#datasets
OGB-Arxiv	RieGrace [35],SSM [39],PDGNNs-SSEM [40],HPNs [2],CGLB [3]	ogb.stanford.edu/docs/nodeprop/#ogbn-arxiv
Reddit	TWP [1],RieGrace [35],DyGRAIN [46],ER-GNN [38],SSM [39],PDGNNs-SSEM [40],SEA-ER [45],CGLB [3]	snap.stanford.edu/graphsage/
OGB-Products	DyGRAIN [46],SSM [39],PDGNNs-SSEM [40],HPNs [2],CGLB [3]	ogb.stanford.edu/docs/nodeprop/#ogbn-products
Cora	RieGrace [35],ER-GNN [38],RLC-CN [42],SGNN-ER [43],SEA-ER [45],HPNs [2]	github.com/tkipf/gcn/tree/master/gcn/data
Citeseer	RieGrace [35],ER-GNN [38],RLC-CN [42],SGNN-ER [43],HPNs [2]	github.com/tkipf/gcn/tree/master/gcn/data
Amazon Computers	TWP [1],RLC-CN [42]	nijianmo.github.io/amazon/index.html
Actor	RieGrace [35],HPNs [2]	github.com/graphdml-uiuc-jlu/geom-gcn/tree/master/new_data/film
Gowalla	GraphSAIL [36],SAER [44]	snap.stanford.edu/data/loc-gowalla.html

be transferable and can benefit the performance on other tasks. By learning transferable features, the catastrophic forgetting problem can also be overcome as a side-effect since the task-wise influence becomes beneficial instead of being detrimental. Works like HPNs [2] and RieGrace [35] have made preliminary attempts to learn transferable features via decomposing each node into basic features or self-supervised learning, respectively. But more works are still required not only for higher performance but also for better explainability of the transferable knowledge.

7.5 Extension to other Modalities

Since graph data is related to various research areas, information from different modalities may accompany the given graphs. For example, images may be attached to products along with its textual name in a co-purchasing network [37]. Therefore, a natural problem is how to incorporate the information from different modalities into the continual learning process. As an early attempt mentioned in Section 4.1, MSCGL [37] approach the problem by accommodating different modalities with separate GNNs. In the following, we list two promising directions for exploring the multi-modal CGL problem. First, during the learning process, information from all modalities may not be available during each task. Therefore, if the modality composition of different tasks are largely different, whether more severe forgetting would be caused and how to avoid it is a practical problem. Second, since each same object (*e.g.* a graph node) is described with information of different modalities, information overlapping may exist across different modalities. Therefore, when trying to preserve the representative data, *e.g.* through a memory buffer, how to avoid storing redundant information and preserve the information efficiently is also practically important.

References

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

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

[3] ——, "Cglb: Benchmark tasks for continual graph learning," in *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022.

[4] G. M. Van de Ven and A. S. Tolias, "Three scenarios for continual learning," *arXiv preprint arXiv:1904.07734*, 2019.

[5] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, "Continual lifelong learning with neural networks: A review," *Neural networks*, vol. 113, pp. 54–71, 2019.

[6] Z. Li and D. Hoiem, "Learning without forgetting," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 12, pp. 2935–2947, 2017.

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

[8] C. Zheng, B. Zong, W. Cheng, D. Song, J. Ni, W. Yu, H. Chen, and W. Wang, "Robust graph representation learning via neural sparsification," in *Proceedings of the 37th International Conference on Machine Learning (ICML)*, July 2020.

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

[10] L. Wang, B. Zong, Q. Ma, W. Cheng, J. Ni, W. Yu, Y. Liu, D. Song, H. Chen, and Y. Fu, "Inductive and unsupervised representation learning on graph structured objects," in *Proceedings of International Conference on Learning Representations (ICLR)*, April 2020.

[11] C. Zhang*, D. Song, C. Huang, A. Swami, and N. V. Chawla, "Heterogeneous graph neural network," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, Anchorage, AK, USA, August 2019, pp. 793–803.

[12] X. Zhang, C. Xu, and D. Tao, "Context aware graph convolution for skeleton-based action recognition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 14 333–14 342.

[13] W. Yu, C. Zhen, W. Cheng, C. Aggarwal, D. Song, B. Zong, H. Chen, and W. Wang, "Learning deep network representations with adversarially regularized autoencoders," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, London, UK, August 2018, pp. 2,663–2,671.

[14] H. Lei, N. Akhtar, and A. Mian, "Spherical kernel for efficient graph convolution on 3d point clouds," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[15] C. Zheng, B. Zong, W. Cheng, D. Song, J. Ni, W. Yu, H. Chen, and W. Wang, "Node classification in temporal graphs through stochastic sparsification and temporal structural convolution," in *Proceedings of European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD)*, September 2020.

[16] L. Lü and T. Zhou, "Link prediction in complex networks: A survey," *Physica A: statistical mechanics and its applications*, vol. 390, no. 6, pp. 1150–1170, 2011.

[17] C. Zhang, H. Yao, L. Yu, C. Huang, D. Song, M. Jiang, H. Chen,

and N. V. Chawla, “Inductive contextual relation learning for personalization,” *ACM Transactions on Information Systems (TOIS)*, vol. 39, no. 35, pp. 1–22, 2021.

[18] L. Cai, J. Li, J. Wang, and S. Ji, “Line graph neural networks for link prediction,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 9, pp. 5103–5113, 2021.

[19] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, “Neural message passing for quantum chemistry,” in *International Conference on Machine Learning*. PMLR, 2017, pp. 1263–1272.

[20] J. T. Vogelstein, W. G. Roncal, R. J. Vogelstein, and C. E. Priebe, “Graph classification using signal-subgraphs: Applications in statistical connectomics,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 7, pp. 1539–1551, 2012.

[21] L. Galke, B. Franke, T. Zielke, and A. Scherp, “Lifelong learning of graph neural networks for open-world node classification,” in *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2021, pp. 1–8.

[22] J. Wang, G. Song, Y. Wu, and L. Wang, “Streaming graph neural networks via continual learning,” in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020, pp. 1515–1524.

[23] Y. Han, S. Karunasekera, and C. Leckie, “Graph neural networks with continual learning for fake news detection from social media,” *arXiv preprint arXiv:2007.03316*, 2020.

[24] W. Yu, W. Cheng, C. C. Aggarwal, K. Zhang, H. Chen, and W. Wang, “Netwalk: A flexible deep embedding approach for anomaly detection in dynamic networks,” in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 2672–2681.

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

[26] L. Zhou, Y. Yang, X. Ren, F. Wu, and Y. Zhuang, “Dynamic network embedding by modeling triadic closure process,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[27] Y. Ma, Z. Guo, Z. Ren, J. Tang, and D. Yin, “Streaming graph neural networks,” in *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2020, pp. 719–728.

[28] Y. Feng, J. Jiang, and Y. Gao, “Incremental learning on growing graphs,” 2020.

[29] P. Bielak, K. Tagowski, M. Falkiewicz, T. Kajdanowicz, and N. V. Chawla, “Fildne: A framework for incremental learning of dynamic networks embeddings,” *Knowledge-Based Systems*, vol. 236, p. 107453, 2022.

[30] F. Zhou, C. Cao, K. Zhang, G. Trajcevski, T. Zhong, and J. Geng, “Meta-gnn: On few-shot node classification in graph meta-learning,” in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 2019, pp. 2357–2360.

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

[32] H. Yao, C. Zhang, Y. Wei, M. Jiang, S. Wang, J. Huang, N. Chawla, and Z. Li, “Graph few-shot learning via knowledge transfer,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 6656–6663.

[33] Z. Tan, K. Ding, R. Guo, and H. Liu, “Graph few-shot class-incremental learning,” in *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, 2022, pp. 987–996.

[34] V. Garcia and J. Bruna, “Few-shot learning with graph neural networks,” *arXiv preprint arXiv:1711.04043*, 2017.

[35] L. Sun, J. Ye, H. Peng, F. Wang, and P. S. Yu, “Self-supervised continual graph learning in adaptive riemannian spaces,” *arXiv preprint arXiv:2211.17068*, 2022.

[36] Y. Xu, Y. Zhang, W. Guo, H. Guo, R. Tang, and M. Coates, “Graphsail: Graph structure aware incremental learning for recommender systems,” in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020, pp. 2861–2868.

[37] J. Cai, X. Wang, C. Guan, Y. Tang, J. Xu, B. Zhong, and W. Zhu, “Multimodal continual graph learning with neural architecture search,” in *Proceedings of the ACM Web Conference 2022*, 2022, pp. 1292–1300.

[38] F. Zhou and C. Cao, “Overcoming catastrophic forgetting in graph neural networks with experience replay,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 5, 2021, pp. 4714–4722.

[39] X. Zhang, D. Song, and D. Tao, “Sparsified subgraph memory for continual graph representation learning,” in *2022 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2022.

[40] X. ZHANG, D. Song, and D. Tao, “Sufficient subgraph embedding memory for continual graph representation learning,” 2022.

[41] D. Wei, Y. Gu, Y. Song, Z. Song, F. Li, and G. Yu, “Incregnn: Incremental graph neural network learning by considering node and parameter importance,” in *Database Systems for Advanced Applications: 27th International Conference, DASFAA 2022, Virtual Event, April 11–14, 2022, Proceedings, Part I*. Springer, 2022, pp. 739–746.

[42] 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*, 2022, pp. 1666–1674.

[43] J. Wang, W. Zhu, G. Song, and L. Wang, “Streaming graph neural networks with generative replay,” in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 1878–1888.

[44] K. Ahrabian, Y. Xu, Y. Zhang, J. Wu, Y. Wang, and M. Coates, “Structure aware experience replay for incremental learning in graph-based recommender systems,” in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2021, pp. 2832–2836.

[45] J. Su and C. Wu, “Towards robust inductive graph incremental learning via experience replay,” *arXiv preprint arXiv:2302.03534*, 2023.

[46] S. Kim, S. Yun, and J. Kang, “Dygrain: An incremental learning framework for dynamic graphs,” in *31st International Joint Conference on Artificial Intelligence, IJCAI 2022*. International Joint Conferences on Artificial Intelligence, 2022, pp. 3157–3163.

[47] A. Daruna, M. Gupta, M. Sridharan, and S. Chernova, “Continual learning of knowledge graph embeddings,” *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1128–1135, 2021.

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

[49] 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)*, 2018, pp. 139–154.

[50] J. Topping, F. Di Giovanni, B. P. Chamberlain, X. Dong, and M. M. Bronstein, “Understanding over-squashing and bottlenecks on graphs via curvature,” *arXiv preprint arXiv:2111.14522*, 2021.

[51] Z. Ye, K. S. Liu, T. Ma, J. Gao, and C. Chen, “Curvature graph network,” in *ICLR*, 2019.

[52] R. Pasunuru and M. Bansal, “Continual and multi-task architecture search,” *arXiv preprint arXiv:1906.05226*, 2019.

[53] D. Lopez-Paz and M. Ranzato, “Gradient episodic memory for continual learning,” in *Advances in Neural Information Processing Systems*, 2017, pp. 6467–6476.

[54] F. R. Hampel, E. M. Ronchetti, P. Rousseeuw, and W. A. Stahel, *Robust statistics: the approach based on influence functions*. Wiley-Interscience; New York, 1986.

[55] P. Buzzega, M. Boschini, A. Porrello, D. Abati, and S. Calderara, “Dark experience for general continual learning: a strong, simple baseline,” *Advances in neural information processing systems*, vol. 33, pp. 15 920–15 930, 2020.

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

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

[58] 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*, 2022, pp. 13 719–13 728.

[59] H. Shin, J. K. Lee, J. Kim, and J. Kim, “Continual learning with deep generative replay,” in *Advances in Neural Information Processing Systems*, 2017, pp. 2990–2999.

[60] A. Chaudhry, P. K. Dokania, T. Ajanthan, and P. H. Torr, “Riemannian walk for incremental learning: Understanding forgetting and intransigence,” in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 532–547.

[61] A. Carta, A. Cossu, F. Errica, and D. Bacciu, "Catastrophic forgetting in deep graph networks: an introductory benchmark for graph classification," *arXiv preprint arXiv:2103.11750*, 2021.
[62] V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, and X. Bresson, "Benchmarking graph neural networks," 2020.
[63] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk, "Slic superpixels compared to state-of-the-art superpixel methods," *IEEE transactions on pattern analysis and machine intelligence*, vol. 34, no. 11, pp. 2274–2282, 2012.
[64] 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," *arXiv preprint arXiv:2005.00687*, 2020.
[65] A. K. McCallum, K. Nigam, J. Rennie, and K. Seymore, "Automating the construction of internet portals with machine learning," *Information Retrieval*, vol. 3, no. 2, pp. 127–163, 2000.
[66] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems*, 2017, pp. 1024–1034.
[67] Z. Wu, B. Ramsundar, E. N. Feinberg, J. Gomes, C. Geniesse, A. S. Pappu, K. Leswing, and V. Pande, "Moleculenet: a benchmark for molecular machine learning," *Chemical science*, vol. 9, no. 2, pp. 513–530, 2018.
[68] Z. Xiong, D. Wang, X. Liu, F. Zhong, X. Wan, X. Li, Z. Li, X. Luo, K. Chen, H. Jiang *et al.*, "Pushing the boundaries of molecular representation for drug discovery with the graph attention mechanism," *Journal of medicinal chemistry*, vol. 63, no. 16, pp. 8749–8760, 2019.
[69] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, "icarl: Incremental classifier and representation learning," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2017, pp. 2001–2010.
[70] A. Chaudhry, M. Rohrbach, M. Elhoseiny, T. Ajanthan, P. K. Dokania, P. H. Torr, and M. Ranzato, "On tiny episodic memories in continual learning," *arXiv preprint arXiv:1902.10486*, 2019.
[71] A. Prabhu, P. H. Torr, and P. K. Dokania, "Gdumb: A simple approach that questions our progress in continual learning," in *European conference on computer vision*. Springer, 2020, pp. 524–540.
[72] J. Yoon, E. Yang, J. Lee, and S. J. Hwang, "Lifelong learning with dynamically expandable networks," *arXiv preprint arXiv:1708.01547*, 2017.