Neural Networks 164 (2023) 156–176

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

2022 Special Issue

Lifelong learning on evolving graphs under the constraints of imbalanced classes and new classes

Lukas Galke [a,*,1], Iacopo Vagliano [b], Benedikt Franke [c], Tobias Zielke [c], Marcel Hoffmann [c], Ansgar Scherp [c]

[a] Max Planck Institute for Psycholinguistics, Nijmegen, Netherlands
[b] Amsterdam UMC, location University of Amsterdam, Netherlands
[c] University of Ulm, Germany

ARTICLE INFO

Article history:
Available online 24 April 2023

Dataset link: https://github.com/lgalke/lifelong-learning, https://doi.org/10.5281/zenodo.3764770

Keywords:
Lifelong learning
Evolving graphs
Graph neural networks
Continual learning
Unseen class detection
Graph representation learning

ABSTRACT

Lifelong graph learning deals with the problem of continually adapting graph neural network (GNN) models to changes in evolving graphs. We address two critical challenges of lifelong graph learning in this work: dealing with new classes and tackling imbalanced class distributions. The combination of these two challenges is particularly relevant since newly emerging classes typically resemble only a tiny fraction of the data, adding to the already skewed class distribution. We make several contributions: First, we show that the amount of unlabeled data does not influence the results, which is an essential prerequisite for lifelong learning on a sequence of tasks. Second, we experiment with different label rates and show that our methods can perform well with only a tiny fraction of annotated nodes. Third, we propose the gDOC method to detect new classes under the constraint of having an imbalanced class distribution. The critical ingredient is a weighted binary cross-entropy loss function to account for the class imbalance. Moreover, we demonstrate combinations of gDOC with various base GNN models such as GraphSAGE, Simplified Graph Convolution, and Graph Attention Networks. Lastly, our k-neighborhood time difference measure provably normalizes the temporal changes across different graph datasets. With extensive experimentation, we find that the proposed gDOC method is consistently better than a naive adaption of DOC to graphs. Specifically, in experiments using the smallest history size, the out-of-distribution detection score of gDOC is 0.09 compared to 0.01 for DOC. Furthermore, gDOC achieves an Open-F1 score, a combined measure of in-distribution classification and out-of-distribution detection, of 0.33 compared to 0.25 of DOC (32% increase).

© 2023 Elsevier Ltd. All rights reserved.

1. Introduction

Graph representation learning has gained momentum in recent years (Hamilton, 2020). Significant developments have been made on graph neural networks (GNNs) based on the seminal work by Scarselli, Gori, Tsoi, Hagenbuchner, and Monfardini (2009) in 2009. In particular, the work on graph convolution (Hamilton, Ying, & Leskovec, 2017; Kipf & Welling, 2017) and graph attention (Veličković et al., 2018) triggered a wave of works that turned GNNs from a niche topic into one of the most active research fields in machine learning (Hamilton, 2020).

The enormous interest in graph representation learning is motivated by the flexibility of graphs to represent virtually any kind of real-world data and the ability to model relationships between data points, i. e., vertices, rather than just the properties of independent and identically distributed (i. i. d.) data points.

A common challenge in machine learning, and thus in graph representation learning, for tasks such as vertex classification is an imbalance in the class distribution. For example, the popular Cora citation dataset (Sen et al., 2008) with seven classes has a heavily skewed class distribution. The smallest class makes about 7% of the vertices, while the largest constitutes about 30%. Citation graphs grow over time. While new publications and citations appear over time, new classes in the form of new scientific fields emerge. In numerous cases, real-world graph data evolves, with new classes, vertices, and edges appearing over time. Generally, this requires the machine learning model to deal with changes and continually adapt the model to new tasks. Adapting a model to new tasks is investigated under the term of lifelong machine learning (Chen & Liu, 2018; Thrun, 1998).

* Corresponding author.
E-mail addresses: lukas.galke@mpi.nl (L. Galke), i.vagliano@amsterdamumc.nl (I. Vagliano), benedikt.franke@uni-ulm.de (B. Franke), tobias-1.zielke@uni-ulm.de (T. Zielke), marcel.hoffmann@uni-ulm.de (M. Hoffmann), ansgar.scherp@uni-ulm.de (A. Scherp).

[1] Parts of this research were carried out while L.G. was with ZBW — Leibniz Information Centre for Economics, Kiel, Germany.

https://doi.org/10.1016/j.neunet.2023.04.022
0893-6080/© 2023 Elsevier Ltd. All rights reserved.

Unsurprisingly, lifelong learning on graph data is also recently gaining more and more interest (Chen, Wang, & Xie, 2021; Febrinanto, Xia, Moore, Thapa, & Aggarwal, 2022; Galke, Franke, Zielke, & Scherp, 2021; Parisi, Kemker, Part, Kanan, & Wermter, 2019; Wang, Qiu, Gao, & Scherer, 2022; Wang, Song, Wu, & Wang, 2020; Zhou & Cao, 2021). Numerous applications can benefit from lifelong graph learning, including social networks, traffic prediction, recommender systems, and anomaly detection (Febrinanto et al., 2022).

Existing works on lifelong graph learning (Cai et al., 2022; Chen et al., 2021; Galke et al., 2021; Wang et al., 2022, 2020; Zhou & Cao, 2021) were mainly concerned with catastrophic forgetting, i. e., how to adapt a model to new data without forgetting what it had learned before. For a recent survey on lifelong graph learning, we refer to Febrinanto et al. (2022). In our prior work, we developed an incremental training procedure to continuously maintain graph representation learning models during the evolution of a graph (Galke et al., 2021). However, the existing body of works, including our own, does not yet address *detection of new classes in the lifelong graph learning scenario*, i. e., dealing with the adaptation of the graph representation under the emergence of new classes while the graph evolves. Generally, when a graph evolves, and new classes emerge, these new classes are relatively rare compared to the number of vertices of already-known classes. Having only a few examples for these new classes further exacerbates the challenge of imbalance in the class distribution because the i. i. d. assumption does not hold for graphs (Hamilton, 2020), particularly when the model continually adapts to changing data. Instead, different influential factors define a vertex label induced by vertex features and the edges between the vertices. Thus, it is exciting to investigate the combination of the two challenges of the imbalanced class distribution and the detection of new classes.

We extend our lifelong training procedure for evolving graphs (Galke et al., 2021) with a new open-world learning module, gDOC, to detect the appearance of new classes in the graph. In particular, we design our gDOC method for detecting new classes to handle imbalanced classes in graph data. It extends the class detection method Deep Open Classification (DOC) (Shu, Xu, & Liu, 2017) from textual data to graph data. We experimentally demonstrate that gDOC can detect new classes in graph data while maintaining high accuracy for classifying in-distribution vertices. The overall performance of gDOC for out-of-distribution (OOD) detection plus vertex classification is consistently higher than a plain DOC. The key to the success of gDOC is weighting the binary cross-entropy loss to counter the imbalanced graph data. Furthermore, we show how to train and retrain graph models to cope with changes, newly emerging classes, and different label rates. We demonstrate that inductively pre-trained graph models are robust to adding unlabeled data. This insight is an essential prerequisite for successful lifelong learning on graph data. For comparability of different temporal datasets, we introduced the k-neighborhood time differences measure (Galke et al., 2021). The measure enables a selection of history sizes in lifelong graph learning that accounts for of the dataset's temporal granularity. We prove that our measure fulfills this critical equivariance property.

1.1. Problem formalization: Lifelong learning on graphs

The critical question in lifelong learning is whether it is helpful to maintain a single model throughout a sequence of tasks versus retraining a new model from scratch for the next task. We call the former case "warm restarts", which means that the initial model parameters for the current task come from the final parameters from the previous task. This reuse of parameter values from the current task to the next task is called *internal knowledge*. The latter case is the "cold restarts" scenario, in which we train a new model from random initialization for each task.

Table 1
Summary of our Notation.

$\boldsymbol{X}$	A matrix holding the vertex features of each vertex as rows
$\boldsymbol{y}$	A vector holding the label of each vertex as its entries
$\mathbb{Y}_t$	The set of classes at time t
$\mathcal{T}$	A task composed of the graph $\mathcal{G}$ along with vertex features $\boldsymbol{X}$ and vertex labels $\boldsymbol{y}$
$\mathcal{T}_t$	Task t within a sequence of tasks
$\mathcal{G}_t$	State of the graph at time t with vertices V_t and edges E_t
c	The history size used for training the GNN
$c(V, E, t)$	A function to determine a history size depending on a set of vertices V and edges E with time information t
$\tilde{\mathcal{G}}_t$	The trimmed graph with respect to the history size c, i. e., older vertices and edges are removed
$\tilde{\boldsymbol{X}}$	A matrix holding vertex features, but with rows removed that correspond to vertices removed in $\tilde{\mathcal{G}}_t$.
$\tilde{\boldsymbol{y}}$	A vector holding vertex labels, but with entries removed that correspond to vertices removed in $\tilde{\mathcal{G}}_t$.

Lifelong learning, i. e., maintaining a single model over time (Thrun, 1998; Thrun & Mitchell, 1995), is only beneficial when warm restarts are at least as good as cold starts under comparable training budgets. In contrast to internal knowledge, there is also *external knowledge*, which is the data used for incremental training. The amount of external knowledge available, i. e., the past graph data, is determined by a *history size*. Note that this history is not separate from the actual data. If the label of a past vertex changes, this change will be reflected in the next incremental training step. The history size is, in turn, determined based on the temporal granularity of the considered graph and the time differences within the receptive field of the GNN. The number of GNN layers defines the receptive field of a GNN. Each layer corresponds to one hop. Thus, the receptive field comprises the k-hop neighborhood of each vertex. Combined, the temporal granularity of the graph and the receptive field allow one to provide comparable results across datasets with different evolution speeds.

We define the problem of new class detection of a graph's vertices in an evolving graph as a form of open-world lifelong graph learning (Chen & Liu, 2018; Galke et al., 2021). We employ this understanding of lifelong graph learning in four different settings. We introduce the four settings below and refer to the corresponding sections for experimental details. Our notation is summarized in Table 1.

Definition 1 (*Lifelong Learning (Chen & Liu, 2018)*)**.** A learner has to perform a possibly open-ended sequence of learning tasks $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_T$. At each time t, the learner is faced with a new learning task $\mathcal{T}_t$, for which it may use the (internal and external) knowledge $\mathcal{K}$ that it has been provided with and accumulated in the previous tasks $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_{t-1}$.

We cast this definition into a lifelong graph learning problem by considering each task $\mathcal{T}_t := (\mathcal{G}_t, \boldsymbol{X}^{(t)}, \boldsymbol{y}^{(t)})$ to be a vertex classification task with graph $\mathcal{G}_t = (V_t, E_t)$, corresponding vertex features $\boldsymbol{X}^{(t)} \in \mathbb{R}^{|V_t| \times D}$, and vertex labels $\boldsymbol{y}^{(t)} \in \mathbb{N}^{|V_t|}$. We denote the set of all classes available at time t as $\mathbb{Y}_t$. We assume that the class distribution $\mathbb{Y}_t$ is skewed and that this distribution changes over time, e. g.,by adding new classes. Thus, any of the vertices that appeared with the graph $\mathcal{G}_t$ may have new, so-far *unseen* classes. This means that $\mathbb{Y}_t$ may contain classes that were not in $\mathbb{Y}_{t-1}$.

To ensure that past knowledge is helpful to perform the task $\mathcal{T}_t$, we impose $\mathcal{G}_{t-1} \cap \mathcal{G}_t \neq \emptyset$. This means there is at least some overlap in the two graphs $\mathcal{G}_{t-1}$ and $\mathcal{G}_t$ of two consecutive tasks.

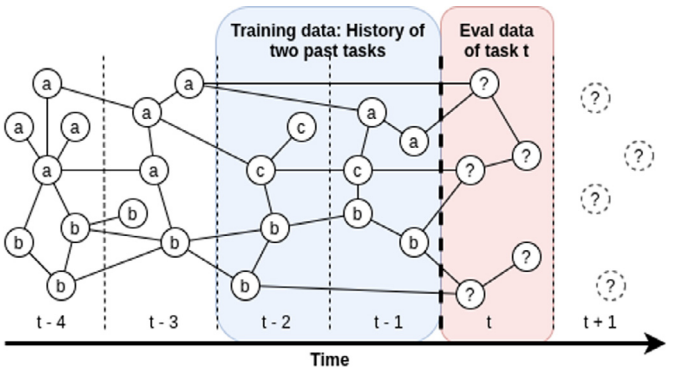

Fig. 1. Illustration of the problem of lifelong graph learning (Galke et al., 2021) with class imbalance and new classes. At each time t, the learner has to classify new vertices of task $\mathcal{T}_t$ (red). Any task might come with previously unseen classes. For example, the class "c" emerged only at task $t-2$ and was subsequently added to the class set. The learner may use internal and external knowledge from previous tasks to adapt to the current task. After evaluating task $\mathcal{T}_t$, we continue with task $\mathcal{T}_{t+1}$.

We assume that the vertices' features and labels do not change after they have appeared, i. e., it holds $\boldsymbol{X}_u^{(t-1)} = \boldsymbol{X}_u^{(t)}, \boldsymbol{y}_u^{(t-1)} = \boldsymbol{y}_u^{(t)}$ if $u \in V_{t-1} \cap V_t$.

In order to control, i. e., limit the amount, of explicit knowledge available for a task $\mathcal{T}_t$, we introduce the history size c. We set $\tilde{\mathcal{T}}_t := (\tilde{\mathcal{G}}_t, \tilde{\boldsymbol{X}}^{(t)}, \tilde{\boldsymbol{y}}^{(t)})$ with $\tilde{\mathcal{G}}_t := \mathcal{G}_t \setminus (\mathcal{G}_1 \cup \mathcal{G}_2 \cup \cdots \cup \mathcal{G}_{t-c-1})$, i. e., we remove vertices (with their features and labels) and edges connecting these vertices that are "older" than $t-c-1$ to construct $\tilde{\boldsymbol{X}}_t$ and $\tilde{\boldsymbol{y}}_t$. Still, the model may use implicit knowledge acquired by the model parameters through warm restarts in earlier tasks for the task $\tilde{\mathcal{T}}_t$.

Based on this formalization, we consider four settings for lifelong learning on graphs: We briefly describe each experimental setting below and refer to the corresponding sections for the details.

Two-task setting. The two-task setting is a simplified setting of lifelong graph learning where we only consider two tasks $\mathcal{T}_1$ and $\mathcal{T}_2$ without new classes, i. e., $\mathbb{Y}_1 = \mathbb{Y}_2$. This setting is suitable for applying our approach to any (non-temporal) standard dataset by defining $\mathcal{T}_1$ as the training graph and $\mathcal{T}_2$ as the test graph. We use this setup to compare transductive and inductive learning on graphs in Section 6. The goal is to test the effect of including unlabeled test data during training, which is relevant for the following settings.

Task-sequence setting. In this setting, one is provided with a sequence of tasks $\tilde{\mathcal{T}}_1, \tilde{\mathcal{T}}_2, \ldots, \tilde{\mathcal{T}}_T$, each with a limited history size. We assume that new classes appear over time, i. e., $\mathbb{Y}_t$ not necessarily equals $\mathbb{Y}_{t-1}$. Although new classes are present in the task $\tilde{\mathcal{T}}_t$'s data in this setting, no methods are employed to detect the new classes. Furthermore, the ground-truth labels $\boldsymbol{y}^{(t-1)}$ to $\boldsymbol{y}^{(t-1-c)}$ are available, when training for the next task $\tilde{\mathcal{T}}_t$. This setting is investigated in the experiments of Section 7.

Task-sequence setting with limited labeled data. This setting is the same as the previous one with the difference that we relax the assumption that all past labels $\tilde{\boldsymbol{y}}$ are available. Here, only a fraction of labels becomes available when training for the next task $\tilde{\mathcal{T}}_t$ instead of the labels of all vertices in the history. This setting is reflected in the experiments of Section 8.

Task-sequence setting with unseen class detection. In the final variant, we analyze the capabilities of the GNN models to detect new classes. In addition to lifelong vertex classification as in previous task-sequence settings, the models now need to emit a binary decision per vertex whether it belongs to a previously known class in $\mathbb{Y}_{t-1}$ (in-distribution), or belongs to a new, unseen class $\mathbb{Y}_t \setminus \mathbb{Y}_{t-1}$ (out-of-distribution). This setting is reflected in the experiments of Section 9.

1.2. Key contributions

We analyze different aspects of lifelong graph learning, detecting new classes and training graph models with skewed class distributions. This research is based on and significantly extends a training procedure and framework for lifelong learning published in our work (Galke et al., 2021; Galke, Vagliano, & Scherp, 2019). The main findings of our prior work are that warm restarts in lifelong learning enable one to use fewer training data, i. e., using a smaller history size. Only a small amount of historical data is necessary to achieve a performance comparable to retraining, i. e., cold start on the entire graph. The key contributions of this work that expand on our previous findings are summarized below.

Method for detecting new classes in lifelong graph learning. We extend our previous lifelong graph learning framework (Galke et al., 2021) with a generic module to detect new classes. For this module, we compare a naive adaption of DOC from text to graphs with a proposed extension, gDOC, that takes into account class imbalance. We find that gDOC outperforms DOC in all cases. Specifically, in the lowest history size setting, gDOC achieves an F1 score of 0.33 compared to 0.25 for DOC (32% increase), and the OOD detection score rises from 0.01 for DOC to 0.09 for gDOC.

Influence of the availability of labels on the performance. We investigate different settings of varying availability of labels in lifelong graph learning: First, we add unlabeled data to the graph after training on the labeled subgraph. We show that adding unlabeled data does not further increase the performance (Galke et al., 2019). In the context of lifelong graph learning, this insight shows that graph models only need to be retrained when new *labeled* becomes available. Second, we vary the label rate between 10% and 90% in a task-sequence lifelong learning setting. We observe a trend that parameter reuse (warm restarts) is generally preferred over cold starts and becomes even more relevant with lower label rates.

The k-neighborhood time difference measure is equivariant to the temporal granularity of evolving graphs. Our k-neighborhood time difference measure $\text{tdiff}_k(\mathcal{G})$ (Galke et al., 2021) captures the temporal differences between connected vertices in evolving graphs. It can be reused independently from the other methods proposed in this work. We use the $\text{tdiff}_k(\mathcal{G})$ measure to determine history sizes comparable between different temporal graphs with differing change behavior (fast versus slow changes). Here, we prove that $\text{tdiff}_k(\mathcal{G})$ is equivariant to temporal granularity, such as when having monthly versus yearly time information.

1.3. Organization of the article

Subsequently, we provide an overview of related work. The extended incremental training algorithm for lifelong graph learning, as well as our new class detection method gDOC, are described in Section 3. In Section 4, we describe the k-neighborhood time differences measure, which we use to determine comparable history sizes across datasets and provide proof that the measure is invariant to different temporal granularities. The datasets used in our experiments are described and analyzed in Section 5. In Sections 6 to 9, we describe the experimental procedure and report the results of our four experiments. We perform experiments for each of the four lifelong graph learning settings introduced in

Section 1.1. First, in Section 6, we analyze the difference between transductive vs. inductive learning, i. e., the influence of adding unlabeled data, on standard (static) datasets, pre-processed in either many-few or few-many train/test splits. Second, we analyze the case of continually adding labeled data during a sequence of tasks in Section 7. Third, in Section 8, we take the most powerful methods and the hardest dataset of the previous experiment to analyze the influence of different label rates. Lastly, in the experiments reported in Section 9, we employ our gDOC method to automatically detect new classes while still being able to correctly classify the vertices of known classes. The results are discussed in Section 10 before we conclude.

2. Related work and selection of models for experiments

Our work connects with various research areas: graph neural networks, lifelong learning, and out-of-distribution detection.

In Section 2.1, we relate our work to the literature on graph neural networks (Hamilton, 2020). Since our aim is that our incremental training algorithm applies to a wide range of GNN models, we seek to obtain high coverage among different types of GNN models in our experiments and select a representative GNN model for each type. In Section 2.2, we relate our work to the lifelong learning literature (Chen & Liu, 2018) concerning general approaches for non-graph data and approaches for graph data. In Section 2.3, we relate our work to the literature on unseen class detection and approaches to the more general problem of out-of-distribution detection while differentiating between supervised and unsupervised, as well as between crisp and scoring approaches.

2.1. Graph neural networks

The success of graph convolution (Kipf & Welling, 2017) has caused a resurgence of interest in graph neural networks (Scarselli et al., 2009). In a generic formulation, the hidden representation of vertex i in layer l is defined as: $h_i^{(l+1)} = \sigma\left(\sum_{j\in\mathcal{N}(i)} \frac{1}{c_{ij}} W^{(l)} h_j^{(l)}\right)$, where $\mathcal{N}(\cdot)$ refers to the set of adjacent vertices and σ is a nonlinear activation function. The normalization factor c_{ij} depends on the respective model: the original Graph Convolutional Networks (GCN) (Kipf & Welling, 2017) use $c_{ij} = \sqrt{|\mathcal{N}(i)|}\cdot\sqrt{|\mathcal{N}(j)|}$.

To categorize the vast literature on graph neural networks, we adopt the distinction of Dwivedi, Joshi, Laurent, Bengio, and Bresson (2020) between isotropic and anisotropic GNN architectures. In isotropic GNNs, all edges are treated equally, while in anisotropic GNNs, the weights for the edges are dynamically calculated, e. g.,based on the features of the involved vertices. Similarly, we differentiate between standard GNN approaches versus scalable approaches that rely on either subgraph sampling or decoupling the neighborhood aggregation from the neural network component. Our goal is to understand how different approaches of GNNs react to situations of evolving graphs and new classes with an imbalanced distribution.

Isotropic graph neural networks. In addition to graph convolutional networks (Kipf & Welling, 2017), examples of isotropic GNNs are GraphSAGE with mean aggregation (Hamilton et al., 2017), DiffPool (Ying et al., 2018), and GIN (Xu, Hu, Leskovec, & Jegelka, 2019). We consider GraphSAGE-Mean (Hamilton et al., 2017) as a representative for isotropic GNNs because its special treatment of the vertices self-connections has been shown to be beneficial (Dwivedi et al., 2020). The representations of self-connections are concatenated with averaged neighbors' representations before multiplying the parameters. In GraphSAGE-Mean, the procedure for obtaining representations on layer $l+1$ for vertex i is given by the equations: $\hat{\boldsymbol{h}}_i^{l+1} = \boldsymbol{h}_i^l \,||\, \frac{1}{\deg_i}\sum_{j\in\mathcal{N}(i)} \boldsymbol{h}_j^l$ and $\boldsymbol{h}_i^{l+1} = \sigma(\boldsymbol{U}^l \hat{\boldsymbol{h}}_i^{l+1})$.

Anisotropic graph neural networks. Examples of anisotropic GNNs include graph attention networks (Veličković et al., 2018), GatedGCN (Bresson & Laurent, 2017), and MoNet (Monti et al., 2017). We consider Graph Attention Networks (GATs) (Veličković et al., 2018) to be representative of the class of anisotropic GNNs. In GATs, the representations in layer $l+1$ for vertex i are computed as follows: $\hat{\boldsymbol{h}}_i^{l+1} = \alpha_{ii}^l \boldsymbol{h}_i^l + \sum_{j\in\mathcal{N}(i)} \alpha_{ij}^l \boldsymbol{h}_j^l$ and $\boldsymbol{h}_i^{l+1} = \sigma(\boldsymbol{U}^l \hat{\boldsymbol{h}}_i^{l+1})$, where $\mathcal{N}(i)$ is the set of adjacent vertices to vertex i, U^l are learnable parameters, and σ is a non-linearity. The edge weights α_{ij} are calculated using a self-attention mechanism based on h_i and h_j, i. e., the softmax of $a(\boldsymbol{U}^l \boldsymbol{h}_i \,||\, \boldsymbol{U}^l \boldsymbol{h}_j)$ on the edges, where a is an MLP and $\cdot\,||\,\cdot$ is the concatenation operation.

Scalable graph neural networks. There are further approaches that have been specifically proposed to scale GNNs to large graphs. These approaches fall into two categories: decoupling neighborhood aggregation from the neural network component (Bojchevski et al., 2020; He et al., 2020; Hu et al., 2021; Rossi, Frasca, et al., 2020; Wu et al., 2019) and subgraph sampling (Chen, Ma, & Xiao, 2018; Chiang et al., 2019; Hamilton et al., 2017; Huang, Zhang, Rong, & Huang, 2018; Zeng, Zhou, Srivastava, Kannan, & Prasanna, 2020).

In simplified GCN (SGC) (Wu et al., 2019), the neighborhood aggregation of GNNs is decoupled from the feature transformation. In SGC, any non-linearities are removed, and consecutive weight matrices are collapsed into a single one. In more detail, SGC can be described by equation $\hat{\mathbf{Y}}_{\text{SGC}} = \text{softmax}(\mathbf{S}^K\mathbf{X}\mathbf{\Theta})$,where $\mathbf{S}$ is the normalized adjacency matrix and $\mathbf{\Theta}$ is the weight matrix. As such, SGC is a scalable variant of Graph Convolutional Networks (Kipf & Welling, 2017) that admits regular minibatch sampling. The hyperparameter K has a similar effect as the number of layers in regular GCNs. Instead of using multiple layers, the k-hop neighborhood is computed by $\mathbf{S}^K$, so that $\mathbf{S}^K\boldsymbol{X}$ can be precomputed. This makes SGC efficient, while, surprisingly, it does not necessarily harm performance (Wu et al., 2019). LightGCN (He et al., 2020) is an approach designed for collaborative filtering that entirely removes the feature transformation and nonlinear activation and only builds upon the neighborhood aggregation of GCNs. Since LightGCN is tailored towards collaborative filtering recommender systems, we opt for SGC in our experiments.

GraphSAINT (Zeng et al., 2020) is a state-of-the-art subgraph sampling technique. In GraphSAINT, entire subgraphs are sampled for training GNNs. Subgraph sampling introduces a bias that is counteracted by normalization coefficients for the loss function. We used the best-performing random-walk sampling for our experiments. The underlying GNN is exchangeable, but the authors suggest using Jumping Knowledge Networks (JKNets) (Xu et al., 2018). JKNets introduce skip connections, or residual connections, to GNNs: Each hidden layer has a direct connection to the output layer, in which the representations are aggregated, for example, by concatenation. FastGCN (Chen, Ma, & Xiao, 2018) is another sampling-based approach, which proposes importance sampling for the assembly of rooted subtree batches. However, GraphSAINT reports a favorable comparison against FastGCN. Thus, we chose GraphSAINT for our experiments.

Dynamic graph methods. Different GNN methods have been proposed for dynamic graphs. An important distinction here is that, in contrast to our problem statement, these methods focus on dealing with varying vertex features and labels over time, e. g.,a user becomes banned from a social network at a specific time (Rossi, Chamberlain, et al., 2020).

This body of work includes dynamic embedding methods (Lee et al., 2021; Nguyen et al., 2018), autoencoder-based methods (Goyal, Chhetri, & Canedo, 2020; Goyal, Kamra, He, & Liu, 2018), GNNs for graphs with a fixed vertex set (Kumar, Zhang,

& Leskovec, 2018; Manessi, Rozza, & Manzo, 2020; Rossi, Chamberlain, et al., 2020; Sankar, Wu, Gou, Zhang, & Yang, 2020; Seo, Defferrard, Vandergheynst, & Bresson, 2018; Trivedi, Dai, Wang, & Song, 2017; Trivedi, Farajtabar, Biswal, & Zha, 2019), and inductive GNN methods that can deal with previously unseen vertices (Da, Chuanwei, Evren, Sushant, & Kannan, 2020; Pareja et al., 2020). These methods focus on the case of dynamic outputs. This means that a vertex can be in class "a" at time t and in class "b" at time $t+1$. In our case, the vertex features and labels remain the same over time, but the graph itself is evolving with new vertices, edges, and classes appearing over time. On the contrary, the related approaches assume a static set of vertices, which makes them inapplicable to the problem of lifelong learning with unseen class detection that we investigate in this paper.

Selection of representative base models. For our lifelong learning experiments, we systematically select representative GNN architectures and scalable GNN techniques. From each of these four categories (anisotropic versus isotropic GNNs, and preprocessing versus sampling), we select one representative for our lifelong graph learning experiments. We chose GraphSAGE as a representative for the class of isotropic GNNs, GAT as a representative for anisotropic GNNs, SGC as a representative for the scaling-via-decoupling approach, and GraphSAINT for the scaling-via-sampling approach. Moreover, we also include JKNets because it is recommended as a base model for GraphSAINT and because its residual connections have been shown to be beneficial (Dwivedi et al., 2020). Lastly, we also include an MLP as a graph-agnostic baseline in all of our experiments.

2.2. Lifelong learning

We first summarize the general literature on lifelong learning, before describing the related work on lifelong learning on graphs.

Lifelong learning on non-graph data. Lifelong learning, or continual learning (Lopez-Paz & Ranzato, 2017), has been present in machine learning research since the mid 1990s (Liu, 2017; Silver, Yang, & Li, 2013; Thrun, 1998; Thrun & Mitchell, 1995). The goal of lifelong learning is to develop approaches that can adapt existing models to new tasks. Although similar on a superficial level, it differs from online learning (Herbster, Pontil, & Wainer, 2005), in which the focus is on processing a data stream efficiently. Ruvolo and Eaton (Ruvolo & Eaton, 2013) introduced a lifelong learning algorithm with convergence guarantees that employs multitask learning so that later tasks can improve earlier tasks. Fei, Wang, and Liu (2016) analyzed SVMs in a lifelong learning environment and introduced cumulative learning. Cumulative learning is related to our approach since we consider that some data are shared among the tasks. Lopez-Paz and Ranzato (2017) introduced a gradient episodic memory framework for the image domain, where examples can be processed independently, and address the catastrophic forgetting problem, i. e., the loss of previously learned information when new information is learned (Robins, 1995).

Similarly to our work, Wang, Chen, Li, and Chen (2021) decompose lifelong learning into the subproblems of rejecting unknown instances, classifying accepted instances, and reducing the cost of learning. However, the work of Wang et al. is on image data, in which the examples are independent of each other, and thus the challenges of dealing with graph data are not reflected. Another promising approach to lifelong learning, and in particular class-incremental learning, is iCaRL (Rebuffi, Kolesnikov, Sperl, & Lampert, 2017), in which prototype vectors of known classes are stored and classification is carried out by taking the nearest distance to these prototypes. However, applying this method to graph data is nontrivial because the vertices are not independent from each other, but connected via edges. For an overview of lifelong learning in general (not specific to graphs), we refer to a recent textbook (Chen & Liu, 2018).

Lifelong learning on evolving graphs. We now focus on the related work on lifelong learning *on graphs*. The challenge of dealing with graph data is a special challenge for lifelong learning approaches. That is because in graphs the nodes are not independent of each other because they are connected through edges. Related work on lifelong learning *on graphs* is still rather limited. We refer to Febrinanto et al. (2022) for a recent survey that covers five recent works on graph lifelong learning.

The most similar approach to ours is Experience Replay GNN (Zhou & Cao, 2021), which proposed to overcome catastrophic forgetting (French, 1999), i. e., the problem of previous knowledge being quickly forgotten when models are adjusted to new tasks. The Replay GNN adapts to new tasks with the help of an experience replay buffer. The buffer holds a subset of the graph that is determined on the basis of different selection strategies: mean of features, coverage maximization, or influence maximization. This work is conceptually similar to our work. However, we use the time information from the nodes in conjunction with a history size to determine which part of the graph is kept in memory.

Wang et al. (2022) proposed a very different strategy to tackle lifelong learning on graphs. Their main goal was again to alleviate catastrophic forgetting. The authors explored a preprocessing step that transforms the vertex classification task into a graph classification task, i. e., each vertex is converted into a feature graph. Therefore, vertices become independent so that they can follow the lifelong learning approach from Lopez-Paz and Ranzato (2017) (see above).

Continual-GNN (Wang et al., 2020) addressed the issue of catastrophic forgetting with a regularization approach. The authors detected new patterns in the data (but not involving any new classes) with an information propagation method. Then they used a combination of experience replay and model regularization to avoid catastrophic forgetting. The result was that their approach leads to performance comparable to model retraining. In relation to this work, we also compare our lifelong-learned models against models retrained from scratch for each task (cold restart) but additionally consider other conditions such as the history size.

Another recent approach (Cai et al., 2022) uses neural architecture search to find a suitable model architecture for lifelong learning on graphs. In particular, the proposed approach focuses on multimodal inputs, such as features extracted via BERT (Devlin, Chang, Lee, & Toutanova, 2019) and vision transformers (Dosovitskiy et al., 2021), rather than dealing with new classes and how to detect them.

Liu, Yang, and Wang (2021) decompose the lifelong learning problem into incremental training over separate tasks, where the class labels are disjunct between tasks (class-incremental). The primary aim is to alleviate catastrophic forgetting. In contrast, we focus on forward transfer, i. e., understanding if previously acquired knowledge is helpful for future tasks given that there is some overlap between classes in the tasks. In addition, we consider the detection of new classes as part of the problem statement.

Tan, Ding, Guo, and Liu (2022) formulate a few-shot class-incremental version of the lifelong learning problem statement and a hierarchical attention-based graph meta-learning approach. The work introduces a regularization objective that aims to avoid overfitting to both, the known classes and new class(es). Their version of the problem statement assumes that some of the new classes' vertices are annotated with a label. In contrast, there are no annotations for the new classes in our problem statement. In other words, we seek to detect vertices that do not belong to any of the known classes, while Tan et al. assume that some labeled information is present such that the few-shot learning setting applies. Both versions have their merits, yet they differ in their

possible use cases: Tan et al. aim to integrate new classes with as few labeled data as possible, while we focus on the problem of automatically detecting new classes.

So far, none of the related works on lifelong learning in graphs considered the problem of detecting unseen classes and rejecting the classification of the respective vertices. Knowing when a model is likely to make mispredictions is a crucial property for deploying reliable systems in practice, which motivates us to explore the combination of lifelong learning on graphs and unseen class detection. Moreover, labeled data is often not fully available in real-world conditions, which we investigate here because it has not yet been considered in previous work on lifelong graph learning.

2.3. Out-of-distribution and unseen class detection

Unseen class detection, or open-world learning, is considered a subcategory of lifelong learning (Chen & Liu, 2018). Still, more general methods for out-of-distribution (OOD) detection are also related to the problem of detecting unseen classes.

Unsupervised out-of-distribution detection. A key challenge is that softmax activation, often used as the final layer of classification, leads to highly confident mispredictions even when the input data are far from the training distribution. To address this, Liang, Li, and Srikant (2018) resorted to temperature scaling, while Lee, Lee, Lee, and Shin (2018) proposed using the Mahalanobis distance. Macêdo and Ludermir (2021), Macêdo, Ren, Zanchettin, Oliveira, and Ludermir (2021) replaced softmax activation with IsoMax activation based on entropy. However, all these approaches only produce an OOD score and neglect the thresholding problem; i. e., they cannot produce crisp decisions for each vertex whether it belongs to a new class or not.

Supervised out-of-distribution detection. Other approaches rely on explicit outlier data that can be used for supervised training of the outlier module (Dhamija, Günther, & Boult, 2018; Hendrycks, Mazeika, & Dietterich, 2019). This is difficult to apply here because we do not distinguish between out-of-distribution and in-distribution but between previously seen classes and previously unseen classes. When we had appropriate training data for the unseen classes, we could train directly on them rather than considering them as OOD. For a detailed discussion of OOD methods, we refer to recent surveys (Pang, Shen, Cao, & van den Hengel, 2021; Yang, Zhou, Li, & Liu, 2021).

Crisp and unsupervised unseen class detection. We are particularly interested in methods that emit a crisp decision on whether the classification of an instance (a new vertex) should be rejected. In this regard, there are several approaches to detect new classes using classic machine learning methods (Bendale & Boult, 2016; Fei et al., 2016; Masud, Gao, Khan, Han, & Thuraisingham, 2011). For example, Wu, Pan, and Zhu (2020) have used variational graph autoencoders for uncertain vertex representation learning. They generate multiple versions of features and test the certainty of a vertex belonging to a known class.

In Deep Open Classification (DOC) (Shu et al., 2017), the authors proposed a method for the detection of new classes in text categorization. To perform the detection, the final softmax activation of a neural network is replaced by elementwise sigmoid activation. Then, they derived a threshold for unseen class detection by measuring the logits' standard deviation across the training set. Their experiments on datasets with balanced classes indicated that DOC is preferable to OpenMax (Bendale & Boult, 2016) and cbsSVM (Fei et al., 2016).

Xu, Liu, Shu, and Yu (2019) propose L2AC for open-world learning in product classification with text data. The L2AC framework is composed of a ranker and a meta-classifier. The ranker retrieves examples from seen classes, which are fed into the meta-classifier to classify the current example or reject its classification. The meta-classifier consists of a matching layer and an aggregation layer. The one-vs-many matching layer determines the similarity to each known class via the top-k known examples. The aggregation layer, a many-to-one BiLSTM, merges the k similarity values into an OOD score per class. After a final fully-connected layer, the classification rule is similar to the one of Deep Open Classification: reject if the score of all classes falls below a threshold of 0.5, or else assign the class label with the maximum logit. Thus, the class detection of L2AC corresponds to a special case of both DOC and gDOC with a fixed threshold of 0.5 and without risk reduction.

Reusing existing OOD detection methods for graphs with interconnected nodes (non-i.i.d.) and imbalanced class distributions is not straightforward. While technically possible, combining graph neural networks with standard OOD methods is rarely evaluated. Here, we transfer the most promising method that is capable of crisp new class detection, DOC, from text to graphs, along with an extension to account for class imbalance.

2.4. Summary

To summarize, lifelong learning on graphs is a new research topic with only a few previous works. The previous works on lifelong graph learning mainly tackle catastrophic forgetting in class- or data-incremental settings on standard datasets. In contrast, we focus on forward transfer, i. e., whether and how much previous knowledge is helpful for future tasks, and use evolving graph datasets close to assumed applications None of the discussed works analyzes the problem of new class detection on graph data we tackle in this work. Moreover, we analyze the effects of label rate, history size, and parameter reuse in incremental learning on evolving graphs with selected representative GNN base models to obtain a complete picture. Although OOD methods are related to new class detection, dealing with interconnected vertices in graphs with imbalanced class distributions is a new challenge, which we tackle in this work.

3. Lifelong and open-world graph learning

We explore the combination of lifelong learning on evolving graphs and new class detection on evolving graphs with imbalanced class distributions. In the following, we recapitulate the incremental training algorithm (Galke et al., 2021), which we extend by a generic module for unseen class detection. Then, we introduce our gDOC method, an extension of DOC (Shu et al., 2017) for unseen class detection, and show how it can be used in conjunction with incrementally trained graph neural networks.

3.1. Training procedure for lifelong graph learning

Our incremental training algorithm for GNNs is shown in Algorithm 1. We assume to have a sequence of T tasks $\mathcal{T}_1, \ldots, \mathcal{T}_T$ and a model f with parameters θ. Throughout the sequence of tasks, the graph changes in the sense that vertices and edges are inserted and deleted. Crucially, the new vertices can come with new classes that have not been part of the training data before. To address these changes, we use the incremental training procedure from our prior work (Galke et al., 2019) for adapting neural networks to new tasks. As a preparation for task $\mathcal{T}_t$, we retrain f on the labels of $\mathcal{T}_{t-1}$ to obtain $\theta^{(t)}$. Whenever l new classes appear in the training data, we add the corresponding number of parameters to the output layer of $f^{(t)}$. Therefore, we have $|\theta^{(t)}_{\text{output weights}}| = |\theta^{(t-1)}_{\text{output weights}}| + l \cdot d_h$ and $|\theta^{(t)}_{\text{output bias}}| = |\theta^{(t-1)}_{\text{output bias}}| + l$, where d_h is the output layer size.

These parameters that model the new classes are randomly initialized. For the other parameters, we consider two options in our incremental training procedure: *warm restarts* and *cold restarts*. With *cold restarts*, we reinitialize $\theta^{(t)}$ and retrain from scratch. On the contrary, when using *warm restarts*, we initialize the parameters for training on task $\mathcal{T}_t$ with the final parameters of the previous task $\theta^{(t-1)}$. Furthermore, we incorporate a generic module (lines 12–14) for unseen class detection in the incremental training algorithm. This operates on the logits of the final output layer and determines whether the classification of a particular vertex should be rejected because it belongs to a class that was not part of the training data.

Algorithm 1 Incremental training for lifelong graph learning under cold-start vs. warm-start condition (extended from (Galke et al., 2021)) .

Require: Sequence of tasks $\tilde{\mathcal{T}}_0, \cdots, \tilde{\mathcal{T}}_T$, model f with parameters θ, flag for cold or warm restarts **Output:** Predicted labels for new vertices of each task along with decision whether it belongs to a previously known class

1: known_classes $\leftarrow \emptyset$
2: $\theta \leftarrow$ initialize_parameters()
3: **for** $t \leftarrow 1$ to T **do** ▷ Iterate through task indices
4: new_classes $\leftarrow$ set($\tilde{\boldsymbol{y}}^{(t-1)}$) $\setminus$ known_classes
5: **if** new_classes $\neq \emptyset$ **then**
6: $\theta' \leftarrow$ expand_output_layer(θ, |new_classes|)
7: **end if**
8: $\theta' \leftarrow$ initialize_parameters()
9: **if** $t > 1$ **and** do_warm_restart = TRUE **then**
10: $\theta' \leftarrow$ copy_existing_parameters(θ) ▷ Reuse prev. model
11: **end if**
12: $\theta' \leftarrow$ train(θ', $\tilde{\mathcal{G}}_{t-1}$, $\tilde{\boldsymbol{X}}^{(t-1)}$, $\tilde{\boldsymbol{y}}^{(t-1)}$) ▷ Train model on prev. task
13: $\hat{\boldsymbol{y}}^{(t)}_{\text{logits}} \leftarrow$ predict(θ', $\tilde{\mathcal{G}}_t$, $\tilde{\boldsymbol{X}}^{(t)}$) for $V_t \setminus V_{t-1}$ ▷ Predict on new nodes
14: $\boldsymbol{m}\text{ood}^{(t)} =$ unseen_class_detection($\hat{\boldsymbol{y}}^{(t)}_{\text{logits}}$) ▷ OOD-Detection
15: $\hat{\boldsymbol{y}}^{(t)}_{\text{pred},i} = \begin{cases} \texttt{OOD} & \text{if } \boldsymbol{m}\text{ood}, i^{(t)} = \texttt{TRUE} \\ \text{argmax}(\hat{\boldsymbol{y}}_{\text{logits}^{(t)},i}), & \text{otherwise} \end{cases}$
16: known_classes $\leftarrow$ known_classes $\cup$ new_classes
17: $\theta \leftarrow \theta'$
18: **end for**

3.2. Self-detection of new classes using our gDOC method

A successful model for lifelong learning would not only classify new data into known classes but would also detect when an instance belongs to a previously unseen class. We seek to develop a generic method that is not specific to any particular GNN architecture. Thus, we take inspiration from the Deep Open Classification (DOC) (Shu et al., 2017) approach that has been proposed for text classification and transfer it to the graph domain.

The key challenges of transferring the DOC method from text to lifelong learning on graphs are how to deal with non-i.i.d. graph data and how to deal with an imbalanced class distribution. We tackle these challenges by combining DOC with a graph neural network and by weighting the binary cross-entropy loss function with the proportions of the class labels seen during training.

Fig. 2 visualizes how we integrate an OOD detection module, such as DOC, into our lifelong node classification framework. A standard graph neural network emits logits for each vertex, while the OOD detection module predicts whether a vertex is in-distribution (ID) or OOD. If the OOD detector emits OOD, we reject to classify the respective vertex with any of the known classes and assume that a new class has been observed. If the vertex is considered ID, the class label is assigned that corresponds to the maximum logit value.

To facilitate OOD detection, the key idea of DOC is to replace the final softmax activation with element-wise sigmoid activation. Hence, the training objective becomes binary cross-entropy rather than categorical cross-entropy. Then, thresholds on the logit distribution over all known classes are used to determine whether the new example belongs to an already known class or not. Below, we briefly summarize the key risk reduction technique proposed in the original DOC, before we describe our extensions.

Thresholds and risk reduction in DOC. To make a clear decision, a threshold is necessary to determine whether a vertex is considered out of distribution (OOD) at the test time. When the output for all classes falls below the threshold, the classification of that vertex is rejected, i. e., the vertex is considered OOD. Such thresholds can be global or class-specific. A natural choice for a global threshold τ is the inflection point of the sigmoid function, i. e., setting $\tau = 0.5$. However, estimating class-specific thresholds can further reduce the risk of incorrectly rejecting the classification of a known class. A strategy for estimating class-specific thresholds is to consult the standard deviation of logits in the training data (Shu et al., 2017). To determine a threshold τ_i for class i, the risk reduction technique proposed in DOC (Shu et al., 2017) collects all model outputs for instances of class i. For all these outputs $\hat{y} \in [0, 1]$, a mirror point $1 + (1 - \hat{y})$ is created, assuming a Gaussian distribution with mean 1. On this distribution, the standard deviation SD_i is calculated to assign the class-specific threshold $\tau_i := \max\{\tau_{\min}, 1 - \alpha \cdot \text{SD}_i\}$, where α is a scaling factor for the standard deviation and $\tau_{\min}$ is the minimum threshold. For α, the original work suggests a value of 3. The authors use a fixed $\tau_{\min} = 0.5$.

Extension to deal with class imbalance (gDOC). Here, we transfer the DOC method to the graph domain. This comprises changing the base model from a 1D-CNN on text to a GNN operating on graphs, as well as changing the loss function for node classification from categorical cross-entropy to binary cross-entropy. In this way, we can employ the same strategy as the original work on DOC for detecting new classes. Throughout this work, we denote this adaptation from text to graph data as "DOC".

We propose an extension, which we denote as gDOC, to make DOC more suitable for lifelong learning on graphs, where we have to deal with a highly imbalanced class distribution. We use a GNN model to emit the logits and adjust the loss scaling of binary cross-entropy to account for class imbalance, which is inevitable in real-world graph data. This is particularly important for unseen class detection because here the magnitude of all outputs is relevant for the final decision, rather than only their maximum value. In detail, if class i appears n^+ times in the training data, we multiply the loss of output i by the factor $\frac{n-n^+}{n^+}$. This is a standard weighting procedure for binary cross-entropy that increases the loss according to the fraction of positive versus negative examples within the training data (cf. (Aurelio, de Almeida, Castro, & de Pádua Braga, 2019)). We denote this variant as gDOC. Furthermore, our experiments will carefully investigate different values for $\tau_{\min}$, while the original DOC (Shu et al., 2017) used a fixed minimum threshold of $\tau_{\min} = 0.5$. Similarly, we also closely investigate the effect of the risk reduction factor α.

3.3. Summary

We have extended the incremental training algorithm with a generic unseen class detection module. As an unseen class

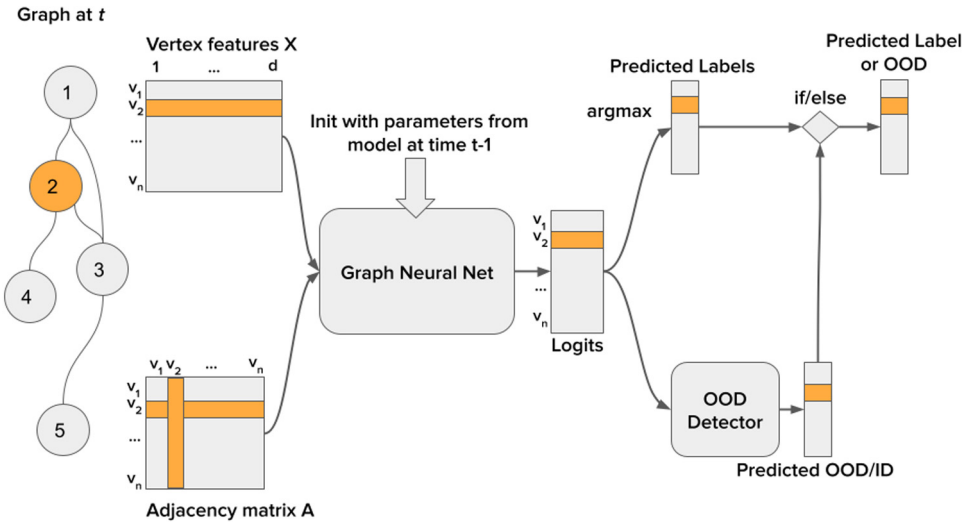

Fig. 2. Procedure of node classification and OOD detection during the execution of a task in lifelong learning. The output logits of the graph neural network are used in two ways. Once to determine the most likely in-distribution class and once to determine whether the example is in-distribution (belongs to a known class) or out-of-distribution. When an example is detected as in-distribution, we return the argmax of the logits. Otherwise, the example is marked as out-of-distribution.

detection module, we introduce gDOC as an extension of the DOC method from text to the challenges of lifelong learning with graph neural networks. Note that both our adaptation of the original DOC to graphs (abbreviated simply as DOC) as well as our extended version (gDOC) can be employed in conjunction with arbitrary GNN base models.

4. Measure of k-neighborhood time differences

Real-world graphs grow and change at different speeds (Aggarwal & Subbian, 2014). Some graphs change quickly, such as social networks, while others evolve rather slowly, such as citation networks. Furthermore, the graphs show different change behavior, i. e., different patterns in how vertices and edges are added and removed over time. Therefore, depending on the specific graph data, a different history of the data must be used for training to take these factors into account. To obtain absolute history sizes that are comparable across different temporal graphs, a measure is needed that provides a history size that is agnostic to the specific change dynamics of the graph (slow vs. fast).

Below, we first introduce such a measure, which we call tdiff_k. In the experiments, we will use the tdiff_k measure to derive candidate history sizes as percentiles of the time differences in the data. Applying the measure can be regarded as a preprocessing step. Subsequently, we show that the history sizes that our measure produces are equivariant to the temporal granularity of the graph.

4.1. Formal definition of the tdiff_k measure

The k-neighborhood Time Difference Distribution measure tdiff_k (Galke et al., 2021) enumerates the distribution of time differences within the k-hop neighborhood of each vertex. This corresponds to the *receptive field* (Chen, Zhu, & Song, 2018) of a GNN with k-many graph convolutional layers. Intuitively, we collect the time differences between all pairs of vertices v and w, which are reachable within at most k edges. We aggregate these time differences based on frequency, i. e., we obtain the number of times a certain time difference has been observed between v and w in the dataset. On this distribution of time differences (represented as a multiset), we compute the percentiles and use them as candidate history sizes.

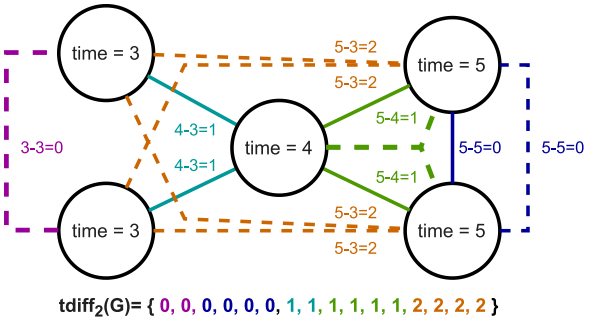

Fig. 3. Example of time differences $\text{tdiff}_2(G)$ for hops at distance of up to 2 from each vertex. Solid lines are edges. Dashed lines indicate paths of length two. Annotations show the time difference between the endpoints of the path. The multiset $\text{tdiff}_2(G)$ holds the resulting time differences. Note that zeros are counted in both directions as both fulfill the $\text{time}(u) \leq \text{time}(v)$ condition.

Definition 2 (*k-Neighborhood Time Difference Distribution*)**.** Given a graph $\mathcal{G} = (V, E)$ and let $\mathcal{N}^k(u)$ be the k-hop neighborhood of vertex $u \in V$ with respect to E, i. e., the set of vertices that are reachable from u by traversing at most k edges. Let time : $V \to \mathbb{N}$ be a function that returns the time information for each vertex v (timestamp metadata), e. g.,the year of publication when considering a citation graph. We define $\text{tdiff}_k(\mathcal{G})$ as *multiset of time differences*, computed over all vertices $u \in V$ to their up to k-distant neighboring vertices $v \in \mathcal{N}^k(u)$ that occurred before vertex u.

$$\text{tdiff}_k(\mathcal{G}) := \{\text{time}(u) - \text{time}(v) \mid \forall u \in V\, \forall v \in \mathcal{N}^k(u) \text{ with } \text{time}(v) \leq \text{time}(u)\}$$

The multiset tdiff_k maps each time difference to the respective number of occurrences and is interpreted as a distribution over time differences. It is used to analyze the temporal distribution of the vertices in a dataset (using percentiles) and to make datasets comparable. Fig. 3 presents an exemplary computation of the k-neighborhood time differences tdiff_2 on a graph with five vertices and five edges. In this example, the 25th percentile of tdiff_2 is 0, the 50th is 1 (also known as median), and the 75th is also 1.

The tdiff_k measure is used in our experiments to compare models trained with a *limited history size* against models trained with the *full history*. Thus, we calculate the 25th, 50th, and 75th

percentiles of the tdiff_k distribution, which we then compare against the full graph (100th percentile) to analyze the influence of explicit knowledge.

4.2. Equivariance to temporal granularity

Any good measure to determine the discrete history sizes $c : (V, E, t) \mapsto \mathbb{N}$ in evolving graphs should be *equivariant to granularity* to ensure comparability between different datasets and different granularities. This means that if we change the perspective, for instance, from years to months, we should get history sizes that are about 12 times larger (on the same data).

More formally, consider two different time measurement functions $t, t' : \mathcal{V} \to \mathbb{N}_{>0}$ whose values differ by a constant factor $a \in \mathbb{R}^+$ such that $t(u) = \lfloor \frac{t'(u)}{a} \rfloor$ for all $u \in \mathcal{V}$. For example, $a = 12$ when comparing the granularities of months t' and years t. In fact, two arbitrary discrete time measurement functions differ by a constant factor with one being coarser-grained (larger denominator) than the other or both being equal ($a = 1$). Then, the derived history sizes should not differ by more than the ratio between the granularity values, i. e., for the measure c to determine the history sizes it should hold that $c(V, E, t') \in [a \cdot c(V, E, t) - a; a \cdot c(V, E, t) + a]$ where $t' \in [a \cdot t - a; a \cdot t + a]$ for all $u \in V$. In the example above, a good measure c should return a history size times 12 plus/minus 12 when we switch the perspective from the year level t to the month level t' (ratio: $a = 12$) on the same data. This property is also crucial for comparable history sizes across datasets with different temporal granularities.

Here, we briefly show that our k-neighborhood time difference measure tdiff_k is equivariant to temporal granularity: We assume without loss of generality that t' is more fine-grained than t, i. e., $a > 1$. Because tdiff_k is a multiset of time differences from which we take percentiles to determine history sizes, it is sufficient to show that the time difference $t(u) - t(v)$ between two vertices $u, v \in V$ is equivariant to the temporal granularity factor a, or more precisely: $\forall u, v \in V : a \cdot |t(u) - t(v)| \in \left]|t'(u) - t'(v)| - a; |t'(u) - t'(v)| + a\right[$, where $]\cdot, \cdot[$ indicates an open interval.

Prerequisite (PRE). With $t(u) = \left\lfloor \frac{t'(u)}{a} \right\rfloor$ we have $\frac{t'(u)}{a} \leq t(u) < \frac{t'(u)+a}{a} \Rightarrow t'(u) \leq a \cdot t(u) < t'(u) + a$. Note that the left-hand side is less than or equal due to rounding down, while the right-hand side adds a time step a, which makes it a true inequality.

Proof. Using the prerequisite, we now show that

(i) $a \cdot |t(u) - t(v)| > |t'(u) - t'(v)| - a$, and

(ii) $a \cdot |t(u) - t(v)| < |t'(u) - t'(v)| + a$

via case differentiation.

Case (i–a): $t(u) = t(v)$ The left-hand side of inequality (i) becomes zero and it remains to show that $|t'(u) - t'(v)| < a$. We apply (PRE) to find the highest possible value for the term $|t'(u) - t'(v)|$ with respect to t such that the term is still smaller than a. The highest possible value for $t'(u)$, expressed in terms of t, is $a \cdot t(u) + \epsilon$ with $0 < \epsilon < a$. This is because $a \cdot t(u)$ is the upper bound of $t'(u)$ in the inequality of the prerequisite (PRE) and we insert a small but positive ϵ to account for "truly lesser". The smallest possible value for $t'(v)$ is $a \cdot t(u)$. Again, we take this value $a \cdot t(u)$ from the prerequisite, where it is the lower bound for $t(u)$. Then, we have $|a \cdot t(u) + a - \epsilon - a \cdot t(v)| < a$. Now, as $t(u) = t(v)$, we obtain $|a - \epsilon| < a$.

Case (i–b): $t(u) \neq t(v)$ We transform the left-hand side of (i) to $|a \cdot t(u) - a \cdot t(v)|$, while recalling that $t \in \mathbb{N}_{>0}$. We use (PRE) to obtain $|t'(u) - t'(v)|$ as the smallest possible value on the left-hand side. Now, the left and right sides are the same except for $-a$ on the right. As $a > 1$ (is positive), inequality (i) is valid.

Case (ii–a): $t(u) = t(v)$ The left-hand side of (ii) becomes zero and it remains to show that $0 < |t'(u) - t'(v)| + a$, which is true because $a > 1$.

Case (ii–b): $t(u) \neq t(v)$ Again, we transform the left-hand side of (ii) to $|a \cdot t(u) - a \cdot t(v)|$. This time, we are interested in the highest possible value with respect to (PRE), which is $|t'(u) + a - \epsilon - t'(v)|$ with $0 < \epsilon < a$. This is because the highest possible difference is between the upper bound $t' + a - \epsilon$ and the lower bound t'. With the triangle inequality, we obtain $|t'(u) + a - \epsilon - t'(v)| \leq |t'(u) - t'(v)| + |a - \epsilon| < |t'(u) - t'(v)| + a$, which holds because $|a - \epsilon| < a$. This concludes the proof. □

4.3. Summary

The k-neighborhood Time Difference Distribution tdiff_k measures the granularity and temporal connectivity patterns of the given graph dataset with vertex-level time information. In general, we can hardly assume that any absolute history size on dataset A would be comparable to the same history size on dataset B. But if we derive the history size from tdiff_k, e. g.,the median of tdiff_2, we have a strategy to find comparable history sizes across datasets, even if they come from different domains, e. g.,social graphs with postings at the minute level versus citation graphs with data on, at least, daily level. This is because our tdiff_k measure is equivariant to granularity and is based solely on time differences between the connected vertices.

5. Datasets and analyses

Adapting models to new data is an important problem whenever machine learning models are deployed in production. However, many graph benchmark datasets are stripped of any temporal data, which is needed to divide the data into realistic partitions for lifelong learning, i. e., a sequence of tasks. To build a lifelong vertex classification dataset with new classes, the following criteria need to be fulfilled:

- attributed vertices,
- vertex labels,
- time information on the vertices,
- *evolving* set of vertices (and thus also edges) over time,
- *evolving* set of classes over time, especially the occurrence of new classes.

We scan the literature (e. g., (Da et al., 2020; Dwivedi et al., 2020; Pareja et al., 2020)) and common dataset collections (OpenGraphBenchmark (Hu et al., 2020), KONECT,[2] and PyTorch Geometric Temporal[3]) for datasets that met the criteria above.

Surprisingly, preprocessed graph datasets that meet these criteria are rare, even though the raw origin of these datasets (social media data, publication metadata) would meet the requirements. In those datasets, in which time information is available, either the graph is static, i. e., it is not an evolving graph, or the set of classes is static, i. e., there are no newly appearing classes over time. Concurrent work on lifelong learning composes the sequence of tasks by synthesizing an ordering of the vertices in static graph datasets (Wang et al., 2022), i. e., data-incremental or class-incremental experimental setups.

In this work, we seek to understand how our methods deal with real-world datasets, i. e., we simulate the evolution of the

[2] http://konect.cc/

[3] https://pytorch-geometric-temporal.readthedocs.io/

Table 2
Statistics for train–test splits: few-many (A) and many-few (B) settings on the citation networks datasets: Cora, Citeseer, and Pubmed. The unseen vertices and edges are available only after the training epochs. The test samples for measuring accuracy are a subset of the unseen vertices. The label rate is the percentage of labeled vertices for training.

Dataset	Cora		Citeseer		Pubmed	
Classes	7		6		3	
Features	1,433		3,703		500	
Vertices	2,708		3,327		19,717	
Edges	5,278		4,552		44,324	
Avg. Degree	3.90		2.77		4.50	
Setting	A	B	A	B	A	B
Train Vertices	440	2,268	620	2,707	560	19,157
Train Edges	342	3,582	139	2,939	34	41,858
Unseen Vertices	2,268	440	2,707	620	19,157	560
Unseen Edges	4,936	1,696	4,413	1,613	44,290	2,466
Test Samples	1,000	440	1,000	620	1,000	560
Label Rate	16.2%	83.8%	18.6%	81.4%	2.8%	97.2%

graph along the time axis and add new vertices and edges according to the time stamps of the vertices. For our first experiment, we used two different splits on standard benchmark datasets with static graphs, which are described next. We can use these datasets to simulate two steps of a temporal graph, where the training data is step one and the *unlabeled* test data is step two. Thereafter, we describe our own temporal datasets that we contribute to the community and use for the other three experiments on lifelong learning.

5.1. Static graph datasets

Cora, Citeseer, and PubMed are standard datasets for the vertex classification task (Sen et al., 2008), which we use for our first experiments on transductive versus inductive learning. The vertices of the graph are research articles represented by textual features and annotated with a class label. Edges are defined by citation relationships but are considered bidirectional. These datasets are often used in transductive learning environments (Kipf & Welling, 2017; Veličković et al., 2018; Yang, Cohen, & Salakhutdinov, 2016). In our experimental setup, we use these datasets to compare inductive vs. transductive learning.

We prepare the static graph datasets in two ways: either a lot of training data and a few test data, or vice versa. Specifically, we used two different train–test splits for each dataset, which we call setting *A* and setting *B*. The setting *A* is derived from the original train–test split for transductive tasks (Kipf & Welling, 2017). It consists of a few labeled vertices that induce our training set and many unlabeled vertices. Setting B instead comprises many training vertices and few test vertices. We set it up by inverting the train–test mask of Setting A and assigning the edges accordingly. Setting B is motivated by applications, in which a large graph is already known and incremental changes occur over time, such as for citation recommendations, link prediction in social networks, and others (Aggarwal & Subbian, 2014; Galke, Mai, Vagliano, & Scherp, 2018). We refer to Table 2 for the details of the datasets and the two settings. We used these three datasets with two different train–test splits in our first experiment described in Section 6.

5.2. Evolving graph datasets

We published three graph datasets for lifelong learning (Galke et al., 2021): one co-authorship graph dataset (PharmaBio) and two DBLP-based citation graph datasets (DBLP-easy and DBLP-hard). For PharmaBio, the classes are journal categories. For DBLP, we use conferences and journals of published papers as classes.

Table 3
Global dataset characteristics: total number of vertices $|V|$, edges $|E|$, features D, and classes $|\mathbb{Y}|$ along with number of newly appearing classes (in braces) within the T evaluation tasks.

Dataset	$\lvert V\rvert$	$\lvert E\rvert$	D	$\lvert\mathbb{Y}\rvert$	T
DBLP-easy	45,407	112,131	2,278	12 (4 new)	12
DBLP-hard	198,675	643,734	4,043	73 (23 new)	12
PharmaBio	68,068	2,1M	4,829	7 (0 new)	18

Since we select those venues with the most publications, this serves as a proxy for a broad categorization. When new conferences and journals emerge, as they do in computer science, new classes appear in the data.

The datasets were generated by imposing a minimum threshold of publications per class per year: 100 for DBLP-easy, 45 for DBLP-hard, and 20 for PharmaBio. For the co-authorship graph PharmaBio, we additionally require a minimum of two publications per author per year. In all datasets, vertex features are normalized TF–IDF representations of the publication title.

5.2.1. Basic characteristics

Table 3 summarizes the basic characteristics of the datasets. DBLP-easy and DBLP-hard are organized into 12 annual snapshots, while PharmaBio has 18 annual snapshots. DBLP-easy has 45k vertices, 112k edges, and a feature dimension of 2,278. The vertices are assigned to one of 12 classes, of which four only appear during the sequence of snapshots, i. e., they are not present in the first snapshots. DBLP-hard has 199k vertices, 644k edges, and a feature dimension of 4,043 (because the word vocabulary is set up based on occurrences within documents). Twenty-three of the 73 classes appear only during subsequent snapshots. PharmaBio comes with 68k vertices, 2.1M edges, feature dimension 4,829, 7 classes, and 18 snapshots. The number of edges is much higher than in the DBLP variants because PharmaBio is a coauthorship graph, which is denser than the citation graphs. Note that DBLP-easy is a subset of DBLP-hard as both were generated by applying a minimum threshold on the number of publications per class.

We report the label distribution of the datasets, the degree distribution, and the distribution over time in Fig. 4. The annual number of publications grows over time. Only in PharmaBio, there is a higher amount of vertices between 1991–1997 than between 1998 and 2003. The global degree distributions of DBLP-easy and DBLP-hard seem to follow a power-law distribution (Newman, 2005) as the degree distribution is almost a straight line except for the blurry tail. For PharmaBio, the degree distribution is more blurry, while a trend line can still be identified. Furthermore, we observe that the number of examples per class is imbalanced in all three datasets. Although the three datasets have different numbers of classes, the shape of the label distributions is similar.

5.2.2. Changes in the class set and distribution shift

Regarding changes in the set of classes, DBLP-easy has 12 venues in total, including one biannual conference and four venues, which appear in 2005, 2006, 2007, and 2012. DBLP-hard has 73 venues, including one discontinued, nine biannual, six irregular venues, and 23 new venues. To quantify the changes in the class set, we calculate the magnitude of the class drift as the total variation distance (Webb, Hyde, Cao, Nguyen, & Petitjean, 2016; Webb, Lee, Goethals, & Petitjean, 2018):

$$\sigma_{t-1,t} = \frac{1}{2} \sum_{y \in \mathbb{Y}_{t-1} \cup \mathbb{Y}_t} |P_{t-1}(y) - P_t(y)|$$

where $P_t(y)$ is the observed class probability at time t. We visualize the drift magnitudes per dataset in Fig. 5. An IID dataset

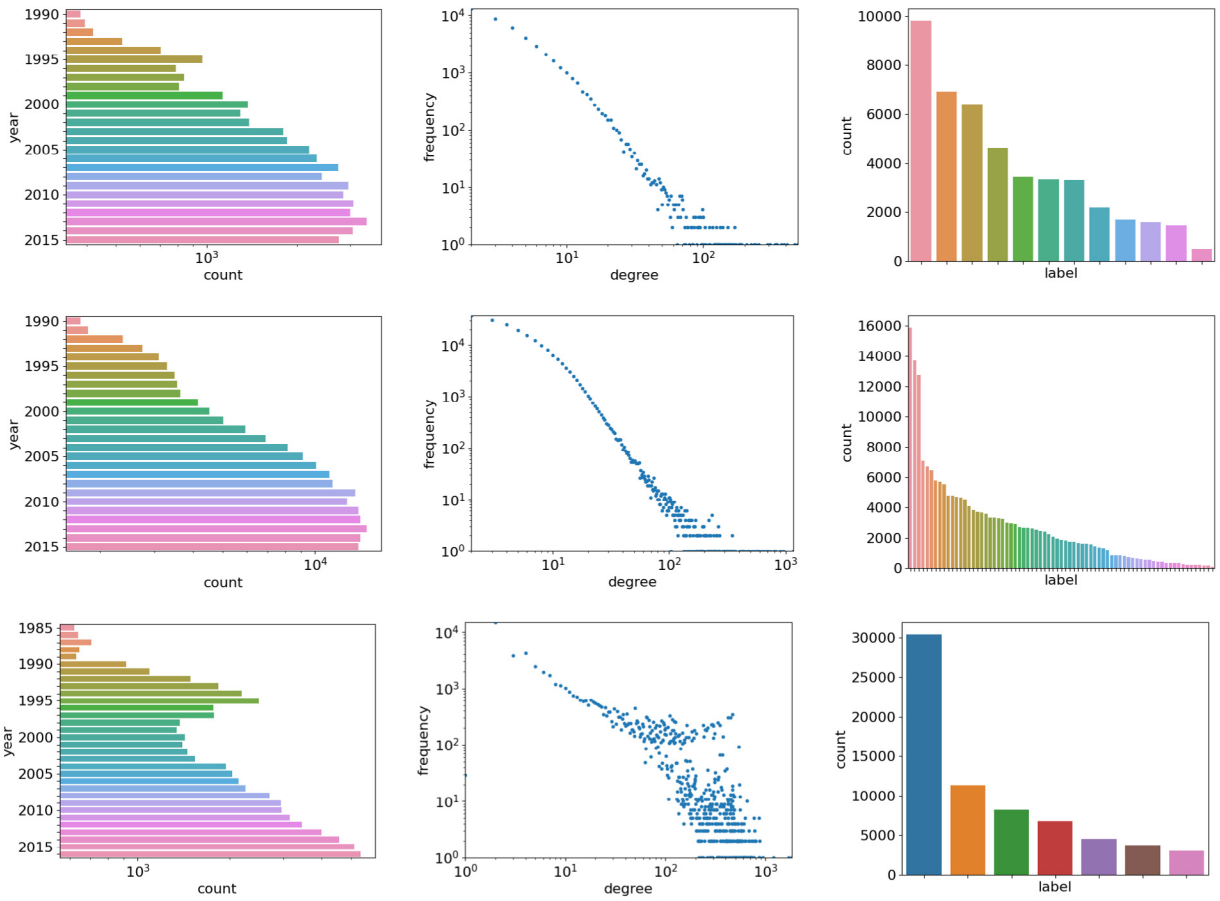

Fig. 4. Distribution of vertices per year on log scale (left column), degree distributions (middle column), label distributions (right column), for our new datasets: DBLP-easy (top row), DBLP-hard (middle row), PharmaBio (bottom row).

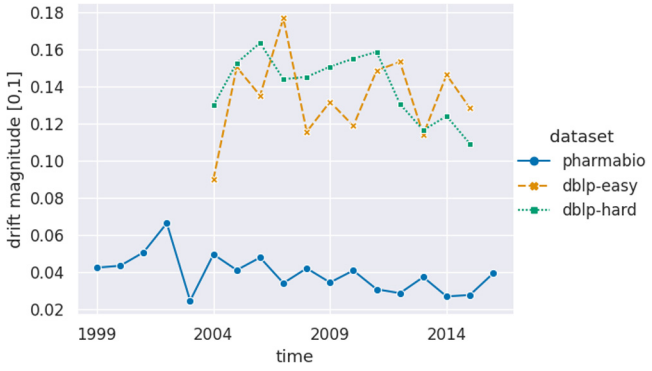

Fig. 5. Magnitude of the class drift per dataset. The drift within the PharmaBio dataset (no new classes) is lower than the drift of both DBLP variants. Independent and identically distributed data would have drift magnitude zero.

would have a drift magnitude of zero by definition. As expected, the drift magnitude is high (between 0.12 and 0.16) for the two datasets with new classes: DBLP-easy and DBLP-hard. On PharmaBio, which does not have new classes, the drift magnitude is consistently lower than 0.07.

5.2.3. Analyzing time differences using tdiff_k

We analyze each dataset using our k-neighborhood time differences tdiff_k introduced in Section 4. In Fig. 6, we show the distributions for three different values of $k = 1, 2, 3$. As expected, the time differences increase if we allow a longer maximum path length k. For our experiments, we will use GNN models with 2 layers, i. e., which take into account the two-hop neighborhood of each vertex. Thus, we use tdiff_2 to derive candidate history sizes, which we will compare to each other in the experiments. Following the distributions for $k = 2$ depicted in Fig. 6, we select 1, 3, 6, and 25 as history sizes for DBLP-{easy,hard} and 1, 4, 8, and 21 as history sizes for PharmaBio according to the 25th, 50th, 75th, and 100th percentiles of tdiff_2.

5.2.4. Dataset preprocessing

For each dynamic dataset, we construct the sequence of tasks $\tilde{\mathcal{T}}_1, \ldots, \tilde{\mathcal{T}}_T$ based on the publication year along with a history size c. For each task $\tilde{\mathcal{T}}_t$, we construct a graph with publications of time $[t - c, t]$, where publications of time t are the test vertices, and $t < c$ training vertices (transductive). We also use it for inductive training, where we train exclusively on $\tilde{\mathcal{T}}_{t-1}$, but still evaluate the test vertices of $\tilde{\mathcal{T}}_t$

We set the first evaluation task $\tilde{\mathcal{T}}_1$ to the time, at which 25% of the total number of publications are available. Therefore, by mapping the datasets to our problem statement (see Fig. 1), our first evaluation task $t = 1$ corresponds to the year 1999 in PharmaBio (1985–2016) and 2004 in DBLP-{easy,hard} (1990–2015). We continue to iterate over the next years for subsequent tasks, i. e., from 2000 to 2016 for PharmaBio and from 2005 to 2015 for DBLP.

5.3. Summary

We have three static graph datasets (Cora, Citeseer, and Pubmed) and three dynamic graph datasets (PharmaBio, DBLP-easy, and DBLP-hard). All datasets have scientific publications

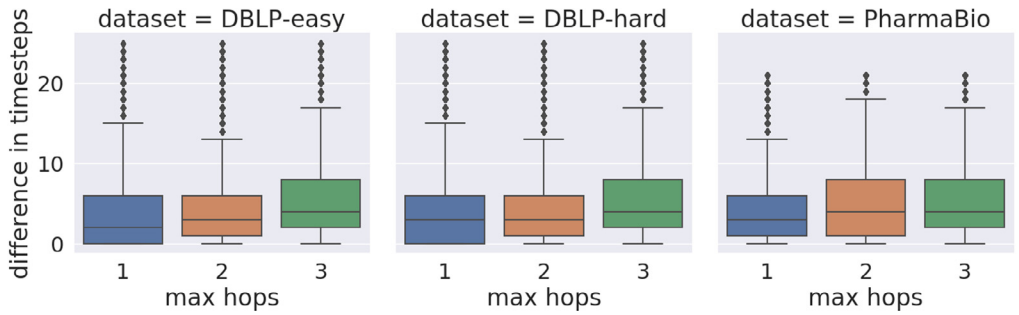

Fig. 6. Distributions of time differences tdiff_k (y-axis) for DBLP-easy (left), DBLP-hard (center) and PharmaBio (right) within the k-hop neighborhood for $k = \{1, 2, 3\}$ (x-axis).

as vertices. All datasets use citations to set up the edges of the graph, except PharmaBio, where the edges are determined by coauthorships. All graph datasets have a highly imbalanced label distribution (see Fig. 4). Two of the dynamic graph datasets come with new classes: DBLP-easy and DBLP-hard, which is reflected in a high class drift over time (see Fig. 5).

We will use the static graph datasets in the first experiments described in Section 6. We will use the dynamic graph datasets in the experiments described in Sections Section 7, 8, and 9.

6. Experiment 1: Transductive versus inductive learning

In the first experiment, our objective is to learn whether accuracy increases when we add unlabeled data to the graph after having trained a model only on the portion of the graph that has labeled vertices. This is important for later experiments because it affects how we move from task t to task $t+1$. We answer whether we need to retrain a model with unlabeled data from the graph at $t+1$, or is it sufficient to wait until the new labeled data become part of the training set. This research question can be very well investigated with the static graph datasets that we introduced in Section 5.1. We use the training set of the static graphs as step one and the unlabeled part of the test set as step two. In order to obtain generalizable results, we consider two different train–test splits for each dataset, which we call setting A (few training, many test examples) and setting B (many training, few test examples), as described in more detail in Section 5.1.

In the context of lifelong learning, settings A and B correspond to different stages of the incremental training procedure. At the very beginning, we start with a few labeled data. After a few tasks, the number of labeled vertices increases, and, then, any new data added to the training set will make only a smaller fraction of the already known labeled data.

In the following, we describe the procedure, hyperparameter, and metrics of our experiments to analyze transductive vs. inductive learning on standard benchmark datasets with two complementary train–test splits. The aim is to analyze the effect of adding unlabeled data after (pre-)training and comparing inductively pre-trained models to models that have been trained transductively *including* the unlabeled test data. We will show that the addition of unlabeled data does not further improve the performance of the inductively pre-trained models.

6.1. Procedure

We construct a dedicated experimental setup to assess the inference capabilities of graph neural networks. We include edges in the training set if and only if its source and destination vertex are both in the training set. The training process is then divided into two steps. First, we pre-train the model on the labeled training set. Then we insert the previously unseen vertices and edges into the graph and continue training for a limited number of inference epochs. The unseen vertices do not introduce any new labels. Instead, the unseen vertices provide features and may be connected to known labeled vertices. We evaluate the accuracy on the test vertices, which form a subset of the unseen vertices, before the first and after each inference epoch. We consider the graph neural networks GCN, GAT, GraphSAGE, as discussed in Section 2, along with a baseline model MLP. For each model, we compare 200 pre-training epochs versus no pre-training. In the latter case, training begins during inference, which is equivalent to retraining from scratch whenever new vertices and edges are inserted. This allows us to assess whether pre-training is helpful for applying graph neural networks on dynamic graphs.

6.2. Hyperparameters

All employed graph neural networks use two graph convolution layers that aggregate neighbor representations. The output dimension of the second layer corresponds to the number of classes. Thus, the features within the two-hop neighborhood of each labeled vertex are taken into account for its prediction. We adopt the same hyperparameter values as proposed in the respective original work. For GCN, we use 16 or 64 hidden units (denoted GCN-64) per layer, ReLU activation, 0.5 dropout rate, along with an (initial) learning rate of 0.005 and weight decay $5 \cdot 10^{-4}$ (Kipf & Welling, 2017). For GAT, we use 8 hidden units per layer and 8 attention heads in the first layer. The second layer has 1 attention head (8 on Pubmed). We set the learning rate to 0.005 (0.01 on Pubmed) with weight decay 0.0005 (0.001 on Pubmed) (Veličković et al., 2018). For GraphSAGE, we use 64 hidden units per layer with mean aggregation, ReLU activation, and a dropout rate of 0.5. We set the learning rate to 0.01 with weight decay $5 \cdot 10^{-4}$ (Hamilton et al., 2017). Our MLP baseline has one hidden layer with 64 hidden units, ReLU activation, a dropout rate of 0.5, a learning rate of 0.005 and a weight decay of $5 \cdot 10^{-4}$. In all cases, we use Glorot initialization (Glorot & Bengio, 2010) and Adam (Kingma & Ba, 2015) to optimize cross-entropy. We initialize the optimizer at the beginning of the inference epochs.

6.3. Measures

Accuracy. We train each model for 35 epochs and repeat the training 100 times with different seeds. The plot shows the mean accuracy plus the standard deviation of the models at each of these training epochs.

Jenson–Shannon divergence. We further compute the Jenson–Shannon divergence (Lin, 1991) on the accuracy distributions to quantify the similarity of the distributions in the two different pre-training configurations (with or without) and in the two different settings (A and B). Since the two distributions are of the same kind, we use a symmetric measure to compare them.

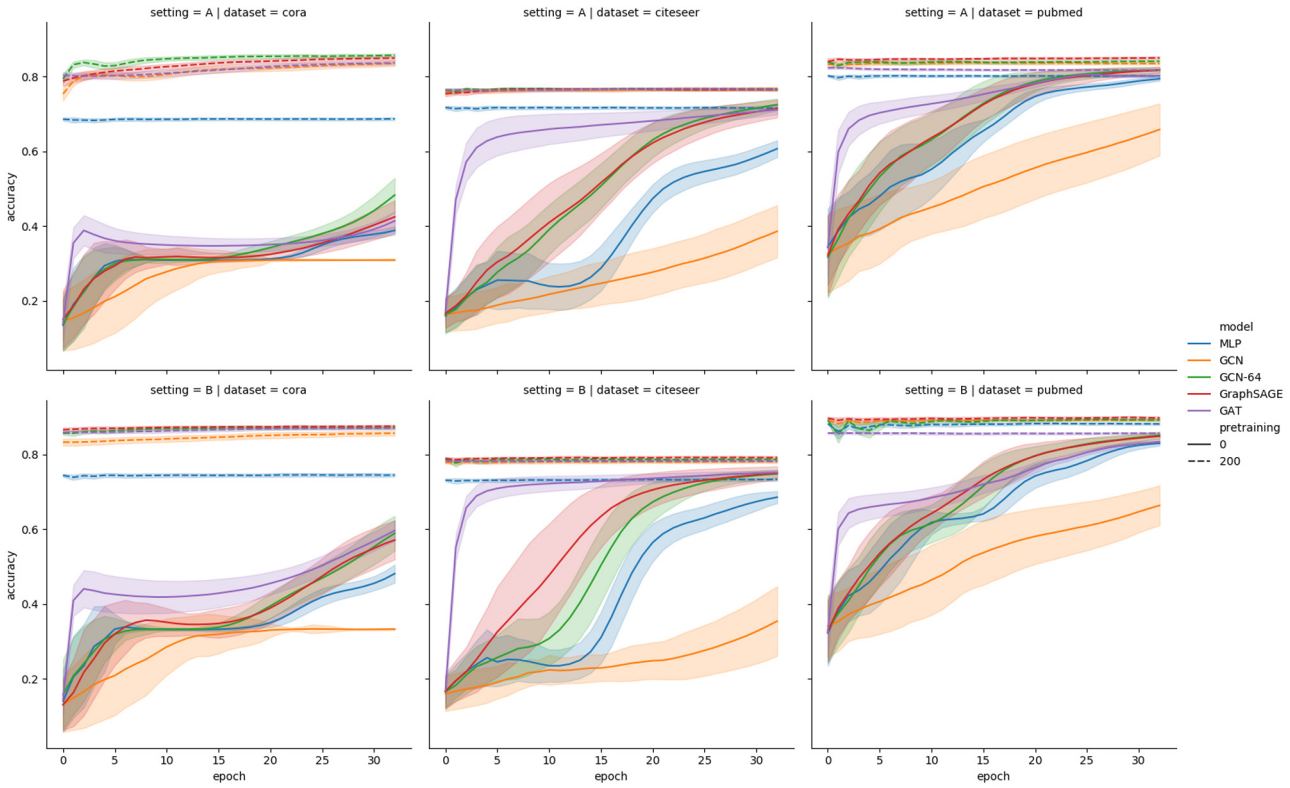

Fig. 7. Test accuracy after each inference epoch for the many-few settings A *Top* and few-many setting B *Bottom* on the datasets Cora, Citeseer, and Pubmed. Each line resembles the mean of 100 runs and its region shows the standard deviation. The dashed lines show the results with 200 pre-training. The solid lines are the results without pre-training.

The Jenson–Shannon divergence (D_{JS}) is such a symmetric measure. It compares two distributions P and Q by calculating the (asymmetric) Kullback–Leibler divergence (D_{KL}) in both directions:

$$D_{\text{JS}}(P \parallel Q) = \frac{1}{2} D_{\text{KL}}(P \parallel Q) + \frac{1}{2} D_{\text{KL}}(Q \parallel P)$$

As D_{JS} is a divergence measure, lower values indicate more similar distributions.

6.4. Results

Fig. 7 shows the results of the GNN models and the MLP on the three datasets: Cora, Citeseer, and Pubmed. The scores of the many-few setting B are higher than those of the few-many setting A by a constant margin. Pre-trained models score consistently higher than non-pre-trained models while having less variance. The accuracy of the pre-trained models plateaus after a few inference epochs (up to 10 on Cora-A, i. e., the Cora dataset investigated in setting A, and Pubmed-B, i. e., setting B applied on the Pubmed dataset). Without any pre-training, GAT shows the fastest learning process. The absolute scores of pre-trained graph neural networks are higher than the ones of MLP. From a broad perspective, the scores of pre-trained graph neural networks are all on the same level. While GCN falls behind the others on Cora-B, GAT falls behind the others on Pubmed in both settings.

We compare the results of setting A and B by measuring the Jensen–Shannon divergence between the accuracy distributions. The Jenson–Shannon divergence between the two settings is lower with pre-training (between 0.0057 for GAT and 0.0115 for MLP) than it is without pre-training (between 0.0666 for GraphSAGE and 0.1013 for GCN). This shows that the accuracy distributions are similar in both train–test splits.

6.5. Summary

Our results show that inductive graph neural networks perform well even though we insert new *unlabeled* vertices and edges after training. For all three datasets, the accuracy plateaus after very few inference epochs. This observation holds for both train–test split settings A and B, i. e., many-few and few-many data for training and testing, respectively. In different terms, we have not observed any gain from up-training an inductive model on extra *unlabeled* data. This motivates us to use the warm restart strategy, i. e., reusing previous parameters, in the following experiments on lifelong learning.

7. Experiment 2: Lifelong learning on graphs

From the previous experiment, we know that inductively trained models are stable when adding unlabeled data after training. Now, we focus on the case in which we continually add more labeled data to the graph, even including new classes in addition to new vertices and edges. The aim is to determine whether parameter reuse is helpful. We consider this question in the context of whether and how many old vertices (and their edges) can be discarded when dealing with evolving graphs.

The challenge in this experiment is that the GNN models have to sequentially adapt to new tasks with new labeled data including unseen classes. We apply the GNNs GraphSAGE, GAT, SGC, GraphSAINT, JKNet, and the baseline MLP on our evolving graph datasets, which we described in Section 5.2. As we know from our analyses of Section 5, the dynamic datasets are naturally heavily imbalanced. The datasets also feature new classes that appear over time. The first appearance of a new class is always at test time, and, only afterward, the vertices with new classes are only added to the training data. In summary, we find interactions

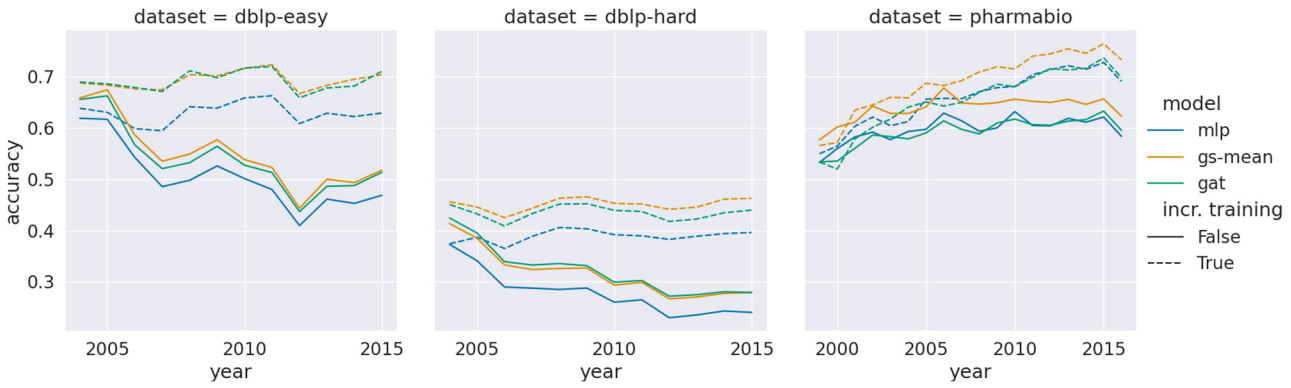

Fig. 8. Results of the ablation study: Accuracy scores of once-trained, static models (solid lines) are lower than incrementally trained models (dashed lines).

between implicit and explicit knowledge: Reusing past parameters (warm restart) enables using smaller history sizes with only a small decrease in performance.

7.1. Procedure

The evolving graph is divided into tasks according to the time slices in years (see Section 5.2). We apply the incremental training algorithm of Section 3.1 to each of the considered models, GraphSAGE, GAT, SGC, GraphSAINT, JKNet, along with a graph-agnostic MLP baseline. The rationale for the selection of these particular base GNN models is provided in Section 2.1.

For each model, we distinguish between warm restart and cold restart configurations, which determines whether the previous parameters are reused as initialization for the next task (warm restart) or not (cold restart).

Furthermore, we consider the history size as a controlled parameter and vary it according to the percentiles of tdiff_k, as determined in our analyses of the datasets in Section 5.2. Corresponding to two layers of graph convolution, which our models use, the quartiles consider 25%, 50%, and 75% of the tdiff_2 distribution and are in terms of history sizes $c = 1$, 3, and 6 for both DBLP datasets, and 1, 4, and 8 for the PharmaBio dataset. We compare these limited-history settings with full-graph training, which corresponds to keeping an unlimited history of the entire timeline of the graph.

All methods are trained in a transductive fashion, except for GraphSAINT, which needed to use the inductive setting. However, we have ensured that the evaluation is fair (see Section 5) and we have confirmed in Experiment 1 (see Section 6) that the difference between inductive and transductive training is negligible.

7.2. Hyperparameters

We constrain all models to two graph convolutional layers, a comparable penultimate hidden dimension (2×32 GraphSAGE, 4×8 GAT, $2 \times 2 \times 16$ JKNet, 64 MLP), and a dropout rate of 0.5. We fix an update step budget of 200 per task and use Kingma and Ba (2015) to optimize cross-entropy. We implemented GAT, GraphSAGE-mean, SGC, and JKNet with *dgl* (Wang et al., 2019) and use *torch-geometric* (Fey & Lenssen, 2019) for GraphSAINT. We had to disable GraphSAINT's norm recomputation for each task so that our experiments could finish in a reasonable time.

For each combination of a GNN, history size, and restart configuration, we tune the learning rate on DBLP-easy. Thus, we consider DBLP-easy as our development dataset to tune the learning rate, which we then apply to DBLP-hard and PharmaBio. The search space for the learning rate is {0.1, 0.05, 0.01, 0.005, 0.001, 0.0005}. We also optimized the weight decay, whose effect was negligible.

For the sake of a fair comparison, we have optimized the hyperparameters separately for each possible history size and restart configuration.

7.3. Measures

Our primary evaluation measure for lifelong vertex classification f is accuracy. With $\text{acc}_t(f^{(t)})$, we denote the accuracy of model $f^{(t)}$ on task $\mathcal{T}_t$. We aggregate the accuracy scores over the sequence of tasks $\mathcal{T}_1, \ldots, \mathcal{T}_T$ by using their unweighted average (Lopez-Paz & Ranzato, 2017):

$$\text{acc}(f) = \frac{1}{T} \sum_{t \in 1,\ldots,T} \text{acc}_t(f^{(t)})$$

Following Lopez-Paz and Ranzato (2017), we use Forward Transfer (FWT) to quantify the effect of reusing previous parameters. This is reflected in the accumulated differences in accuracy between the f_{warm} and f_{cold} models, defined below:

$$\text{FWT}(f_{\text{warm}}, f_{\text{cold}}) = \frac{1}{T-1} \sum_{t \in 2,\ldots,T} \text{acc}_t(f^{(t)}_{\text{warm}}) - \text{acc}_t(f^{(t)}_{\text{cold}})$$

Experiments are repeated 10 times with different random seeds. We report the mean accuracy plus/minus 1.96 times the standard error of the mean.

7.4. Results

Table 4 shows the aggregated results of 20,160 evaluation steps (48 configurations with 10 repetitions on two datasets with 12 tasks each and one dataset with 18 tasks). We consider the method A to be better than the method B when the mean accuracy of A is higher than that of B and the 95% confidence intervals do not overlap (Goodfellow, Bengio, & Courville, 2016). In terms of the absolute best methods per setting (=dataset $\times$ history size), we find that GraphSAGE consistently gives the highest scores except for DBLP-hard, where it is challenged by SGC.

Regarding the comparison of history sizes (i. e., *explicit knowledge*, see Section 1), the highest scores are achieved in almost all cases by using an unlimited history size, i. e., using the full graph's history. However, in all datasets, the scores for training with limited window sizes larger than 1 are close to those for full-graph training. With history sizes that cover 50% of the GNN's receptive field, all methods achieve at least 95% relative accuracy compared to the same model under full-history training. When 75% of the receptive field is covered, the models produce at least 99% relative accuracy. To compute these percentages, we have selected the best of both cold and warm restarts for each method.

Table 4
Accuracy (with 95% confidence intervals through 1.96 standard error of the mean) and Forward Transfer (averaged difference of warm and cold restarts) in our datasets with different history sizes (column **c**). The best method per case (= per one dataset and one history size) is marked in bold, along with the methods where the 95% CI overlaps.

Dataset	c	GAT			GraphSAGE-Mean			MLP (Baseline)		
		Avg. accuracy		FWT	Avg. accuracy		FWT	Avg. accuracy		FWT
		cold	warm		cold	warm		cold	warm	
DBLP-easy	1	60.8 ± 0.5	$\mathbf{64.9 \pm 0.4}$	+4.5	60.4 ± 0.5	$\mathbf{65.1 \pm 0.4}$	+5.2	56.1 ± 0.4	62.2 ± 0.5	+6.6
	3	$\mathbf{68.9 \pm 0.3}$	$\mathbf{69.3 \pm 0.3}$	+0.2	$\mathbf{68.7 \pm 0.3}$	$\mathbf{69.3 \pm 0.3}$	+0.7	61.0 ± 0.5	62.9 ± 0.4	+2.0
	6	$\mathbf{70.3 \pm 0.4}$	70.2 ± 0.4	−0.1	$\mathbf{71.1 \pm 0.4}$	$\mathbf{70.9 \pm 0.4}$	−0.3	62.7 ± 0.3	62.7 ± 0.4	−0.2
	full	70.2 ± 0.4	70.2 ± 0.4	+0.1	$\mathbf{71.6 \pm 0.4}$	$\mathbf{71.4 \pm 0.3}$	−0.2	63.4 ± 0.3	61.9 ± 0.4	−1.2
DBLP-hard	1	39.4 ± 0.2	39.1 ± 0.2	−0.1	34.5 ± 0.4	40.0 ± 0.2	+5.9	31.6 ± 0.3	38.3 ± 0.3	+7.4
	3	44.0 ± 0.2	43.7 ± 0.2	−0.4	44.3 ± 0.2	$\mathbf{45.1 \pm 0.2}$	+0.8	33.7 ± 0.3	38.9 ± 0.2	+5.6
	6	45.1 ± 0.3	45.3 ± 0.3	+0.2	$\mathbf{46.5 \pm 0.3}$	$\mathbf{46.7 \pm 0.3}$	+0.2	39.2 ± 0.2	38.3 ± 0.2	−0.7
	full	45.6 ± 0.3	45.6 ± 0.3	−0.1	46.8 ± 0.2	47.1 ± 0.3	+0.4	38.2 ± 0.2	36.7 ± 0.2	−1.1
PharmaBio	1	61.6 ± 0.9	65.4 ± 0.9	+3.8	65.4 ± 0.9	$\mathbf{68.6 \pm 1.0}$	+3.3	62.7 ± 0.9	66.3 ± 0.9	+3.9
	4	64.5 ± 0.8	65.3 ± 0.9	+0.9	$\mathbf{68.0 \pm 0.8}$	$\mathbf{69.0 \pm 0.8}$	+1.1	66.3 ± 0.7	65.7 ± 0.8	−0.7
	8	65.1 ± 0.8	65.4 ± 0.8	+0.3	$\mathbf{68.8 \pm 0.7}$	$\mathbf{69.0 \pm 0.8}$	+0.2	64.2 ± 0.8	65.3 ± 0.7	+0.9
	full	64.3 ± 0.8	65.4 ± 0.8	+0.2	$\mathbf{69.0 \pm 0.7}$	$\mathbf{68.4 \pm 0.7}$	−0.7	65.4 ± 0.8	64.4 ± 0.6	−1.1
		SGC			GraphSAINT			Jumping Knowledge		
		Avg. accuracy		FWT	Avg. accuracy		FWT	Avg. accuracy		FWT
		cold	warm		cold	warm		cold	warm	
DBLP-easy	1	57.1 ± 0.4	63.7 ± 0.4	+7.2	62.1 ± 0.3	63.2 ± 0.4	+1.2	56.2 ± 0.5	61.4 ± 0.5	+5.6
	3	66.4 ± 0.3	67.4 ± 0.3	+1.2	66.4 ± 0.4	65.3 ± 0.5	−0.9	65.2 ± 0.3	65.9 ± 0.5	+1.0
	6	69.3 ± 0.4	69.3 ± 0.4	+0.1	68.1 ± 0.4	65.5 ± 0.7	−2.1	68.0 ± 0.4	66.9 ± 0.6	−0.7
	full	$\mathbf{71.0 \pm 0.4}$	70.0 ± 0.4	−1.0	68.4 ± 0.5	65.7 ± 0.5	−2.8	68.7 ± 0.4	66.3 ± 0.4	−2.5
DBLP-hard	1	34.5 ± 0.3	$\mathbf{41.0 \pm 0.3}$	+7.0	35.9 ± 0.3	35.6 ± 0.4	+0.5	33.0 ± 0.2	35.3 ± 0.3	+2.9
	3	44.1 ± 0.2	$\mathbf{44.8 \pm 0.3}$	+0.8	39.3 ± 0.3	38.1 ± 0.5	−0.6	39.1 ± 0.3	38.8 ± 0.4	+0.3
	6	$\mathbf{46.9 \pm 0.3}$	46.2 ± 0.3	−0.4	40.6 ± 0.3	38.8 ± 0.6	−1.2	41.0 ± 0.3	40.1 ± 0.5	−0.3
	full	$\mathbf{48.8 \pm 0.4}$	47.5 ± 0.3	−1.2	41.0 ± 0.4	40.7 ± 0.4	−0.3	41.6 ± 0.3	40.8 ± 0.2	−0.9
PharmaBio	1	62.3 ± 0.9	64.5 ± 0.8	+2.3	65.7 ± 0.8	$\mathbf{68.6 \pm 0.8}$	+3.0	64.1 ± 0.9	$\mathbf{68.3 \pm 0.9}$	+4.3
	4	64.4 ± 0.8	64.4 ± 0.8	−0.0	67.3 ± 0.8	$\mathbf{68.4 \pm 0.7}$	+1.0	67.1 ± 0.8	$\mathbf{68.2 \pm 0.8}$	+1.1
	8	65.3 ± 0.8	64.0 ± 0.7	−1.4	$\mathbf{68.1 \pm 0.8}$	$\mathbf{68.0 \pm 0.7}$	−0.1	$\mathbf{67.8 \pm 0.8}$	$\mathbf{67.7 \pm 0.7}$	−0.3
	full	62.4 ± 0.8	61.7 ± 0.6	−0.8	$\mathbf{68.2 \pm 0.8}$	66.1 ± 0.8	−2.2	66.8 ± 0.8	64.5 ± 0.7	−2.6

Regarding the influence of *implicit knowledge*, we find that reusing parameters (warm restarts) leads to notably higher scores than retraining from scratch when the history size is one (see column FWT with history size $c = 1$). The average Forward Transfer across all models and datasets with history size $c = 1$ is five accuracy points.

Regarding isotropic vs. anisotropic GNNs, we find that GAT and GraphSAGE perform similarly well on DBLP-easy (on which the learning rate was tuned). However, GraphSAGE-mean yields higher scores on DBLP-hard and PharmaBio, which could indicate that GraphSAGE-mean is more robust to hyperparameters than GAT.

Regarding memory-efficient methods, we observe that the scores of SGC are among the highest of all methods on DBLP-hard. To understand this result, we recall that SGC uses only one single weight matrix of shape $n_{\text{features}} \times n_{\text{outputs}}$, which leads to 300,000 learnable parameters on DBLP-hard, but only 27,000 and 34,000 on DBLP-easy and PharmaBio, respectively. SGC maps input features directly to classes, which results in a very high number of parameters on DBLP-Hard because this dataset has a high number of classes. For comparison, GraphSAGE has 146,000 learnable parameters on DBLP-easy, 264,000 on DBLP-hard, and 310,000 on PharmaBio. On the other hand, GraphSAINT yields scores on PharmaBio comparable to GraphSAGE, but lower scores on both DBLP datasets.

7.5. Ablation study: Incrementally-trained vs. once-trained models

In contrast to retraining with different history sizes, one may also wonder how long a once-trained model generalizes. Thus, we have analyzed how long a model, which is trained only once at a specific point in time, will generalize well over a sequence of tasks over time. We isolate the effect of incremental training and compare once-trained trained models (static) with incrementally trained models (incremental). Static models are trained for 400 epochs on the data before the first evaluation time step, which comprises 25% of the total vertices. Incrementally trained models are trained for 200 epochs with history sizes of 3 timesteps (4 on the PharmaBio dataset) before evaluating each task. We repeat each experiment 10 times with different random seeds. In Fig. 8, we see that the accuracy of the static models decreases with time on DBLP-easy and DBLP-hard, where new classes appear over time. On PharmaBio (fixed class set), the accuracy of the static models plateaus, whereas the accuracy of incrementally trained models increases. We see that incremental training is not only necessary to adapt to new classes, but also helpful to make use of an increased amount of training data.

7.6. Summary

This experiment shows that in the three analyzed datasets, with only history sizes of 3 or 4 (corresponding to 50% coverage of the receptive field of a 2-layer GCN), almost all methods obtain 95% accuracy compared to the same model under full-history training. Moreover, with very small history sizes, such as using only one past task, using warm restarts is important to maintain a high level of accuracy. Furthermore, we have confirmed in an ablation study that incremental training is necessary to account for changes of the graph.

8. Experiment 3: Lifelong learning with limited labeled data

Until now, we have assumed that the true labels of vertices become part of the training data for subsequent tasks. Now we relax this assumption and release only a portion of the labeled

data in task t for training in the subsequent task $t + 1$. This resembles real-world applications, such as the indexing of scientific articles in libraries (Mai, Galke, & Scherp, 2018). The motivation is that labeled data is expensive to "produce". Again, we work with the most challenging dataset, DBLP-hard, for this experiment, because it has the highest number of new classes.

8.1. Procedure

To implement the idea of learning with only a fraction of labeled data, we randomly sample a subset of vertices, for which true class labels are available for training. We denote this fraction as *label rate*. For the experiments, it is important to sample globally rather than on a per-task basis to avoid nodes toggling between being labeled and unlabeled. Therefore, we sample the entire dataset before splitting it into tasks. In this way, the subset of vertices that comes with classes is fixed for the entire duration of the experiments. Furthermore, we used the same subset of classes with all different configurations and all repetitions of the experiment. We sample uniformly at random on the vertex level without any stratification between classes. Note that this problem statement of testing different label rates is similar to the difference between settings A and B in Experiment 1 of Section 6. However, here we test the influence of the label rate in the context of a task sequence (instead of comparing only two tasks) and systematically change the label rates ranging from 0.1 to 0.9 (instead of only one "split").

For this experiment, we use GraphSAGE-Mean as the GNN model because it achieved the best results in the previous experiment, where the label rate was not restricted. As above, we experiment with different history sizes 1, 3, and 6 and both restart configurations warm and cold. As the dataset, we use DBLP-hard because it has the highest number of classes both in total and new classes that appear over time, and thus is the most challenging.

8.2. Hyperparameters

Again, as in the previous experiment, the optimal hyperparameters were determined on DBLP-easy, the sub-dataset of DBLP-hard that we consistently use to tune hyperparameters. The search space for the learning rate is again $\{0.1, 0.05, 0.01, 0.005, 0.001, 0.0005\}$. We have not tuned the hyperparameters separately for each label rate, but we reuse the optimal hyperparameters from training with a 100% label rate.

8.3. Measures

As in the previous experiment, we use the average accuracy across tasks as the evaluation metric.

8.4. Results

In Fig. 9, we plot the average accuracy between tasks as a function of the label rate. As expected, the absolute accuracy values decrease as the label rate decreases. However, we made a similar observation as in previous experiments with respect to warm/cold restarts. Using warm restarts consistently leads to higher scores than cold restarts. The effect is more prolonged when the history size is small.

When comparing history sizes, we again observe that a larger history size leads to better results. In particular, using the entire history gives the best results, closely followed by a history size of 6. Still, when the label rate is decreased, the difference between the history sizes remains constant.

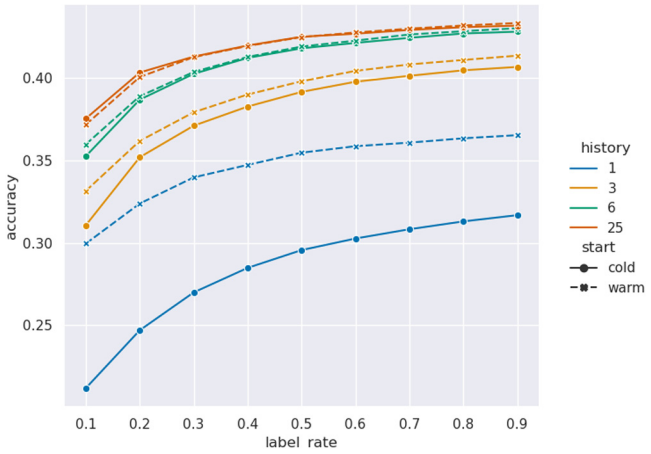

Fig. 9. Average accuracy of GraphSAGE with warm restarts across tasks on DBLP-hard under varying label rate.

With very low label rates (in the range between 10% and 30%), the accuracy of the cold restart strategy drops faster than the accuracy of warm restarts. In other words, the use of warm restarts leads to more stable models when dealing with lower label rates.

8.5. Summary

This experiment shows that the effect of varying the label rate is as expected: the performance degrades with fewer labeled training data. We confirm the finding from previous experiments that warm restarts consistently lead to higher performance than cold restarts when the history size is small. Furthermore, we observe that warm restarts become even more important when the label rate is low.

9. Experiment 4: Detection of unseen classes

In our evolving graphs, we have to deal with previously unseen classes. In previous experiments on lifelong learning, these unseen classes (and the vertices that have these classes) were already part of the test data. However, the models did not have the opportunity to actually predict those classes, as no dedicated technique has been used to detect vertices from unseen classes. Here, we evaluate our adaptation of the unseen class detection method DOC to graph data, called gDOC, as introduced in Section 3.2. The experiments comprise a crisp unsupervised detection of instances of unseen classes. At the same time, the models need to make predictions as usual for the known classes.

9.1. Procedure

In previous experiments, unseen classes were part of the test data, while there was no active treatment of having them detected automatically. In this experiment, we seek to evaluate the performance of the gDOC method to detect unseen classes. As before, we train on task $t - 1$ and evaluate on t over a sequence of T tasks. However, for each vertex, we use our unseen class detection module gDOC to predict whether this vertex belongs to a previously known class or not. If the prediction is that the vertex does not belong to any previously known class, we reject its classification and assign a special virtual class ("unseen"). As unseen class detection modules, we compare the original DOC as a baseline with our proposed gDOC method.

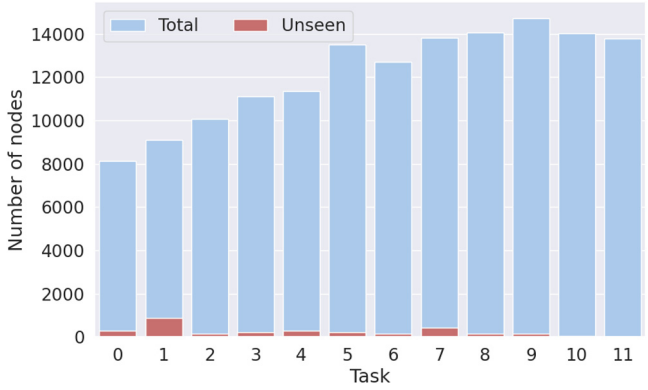

Fig. 10. Number of vertices with unseen classes per task on DBLP-hard.

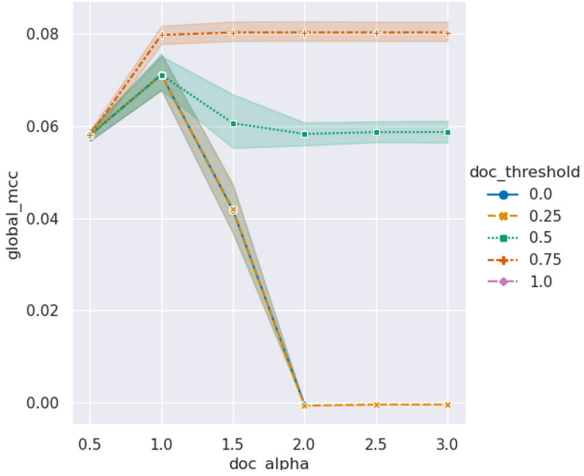

Fig. 11. MCC score of gDOC with GraphSAGE-mean as GNN model (history size 3, warm restart setting) as a function of the risk reduction factor α and varying *minimum* threshold values. We observe that the more risk reduction does not improve the results.

We used the DBLP-hard dataset, which has 23 new classes. In addition to the dataset analysis in Section 5.2, we show in Fig. 10 how many vertices belong to unseen classes in the DBLP-hard dataset. We also experiment with DBLP-easy, which has 4 new classes. We use the best-performing model GraphSAGE-mean along with gDOC for unseen class detection that we have introduced in Section 3.2. Our baseline is the original DOC method, also applied to the outputs of GraphSAGE-mean. We observe that in every task except for the last one, there are vertices with unseen classes.

9.2. Hyperparameters

As in the previous experiments, we optimize the model hyperparameters in our development dataset DBLP-easy. We repeat the hyperparameter optimization because the loss function has changed from categorical to binary cross-entropy. As before, the best learning rate is selected based on the best accuracy on DBLP-easy and transferred to DBLP-hard.

Note that we did not tune the learning rate for unseen class detection performance, but for the best accuracy, as in previous experiments. We then compare DOC with gDOC, where the former is our baseline and the latter uses our proposed class weighting loss function for lifelong learning.

9.3. Measures

We evaluate how well the models detect unseen classes. For this purpose, we use two measures: Macro-F1 with a special class for instances of unseen classes (Shu et al., 2017) and the Matthews correlation coefficient (MCC). Note that Macro-F1 averages the F1 scores over classes such that the effect of the 'unseen' class is taken into account as any of the known classes. In detail, we compute this *Open Macro-F1* as

$$\text{Open Macro-F1} := \frac{1}{T}\sum_{t=1}^{T} \text{Macro-F1}(\boldsymbol{y}'(t), \boldsymbol{y}'_{\text{pred}}(t))$$

with

$$\boldsymbol{y}'_{\text{pred},i} := \begin{cases} \text{'unseen', if example } i \text{ is detected as OOD} \\ \boldsymbol{y}_{\text{pred},i}, \text{ otherwise} \end{cases}$$

$$\boldsymbol{y}'_i := \begin{cases} \boldsymbol{y}_i \text{ if class } \boldsymbol{y}_i \text{ is known} \\ \text{'unseen', otherwise} \end{cases}$$

where $\boldsymbol{y}_i$ are the true labels and $\boldsymbol{y}_{\text{pred}}$ are the predicted class labels. The arg max of the output is replaced by a special symbol when the method has emitted an 'unseen' decision for that instance. The true labels $\boldsymbol{y}$ are preprocessed similarly so that instances of previously unseen classes receive a special class symbol.

In pre-experiments, we found that the best Open F1-Macro scores are achieved when the thresholds are high. This is because we have a high number of classes and the special class contributes only very little to the overall F1 Macro score. Thus, a large number of false rejects, i. e., a reject despite the class being known, diminishes the overall performance in terms of F1-Macro.

The F1-Macro score is limited in its expressiveness with respect to the detection of unseen classes since there are only a few vertices of that unseen class. Thus, we report a further score, the Matthews correlation coefficient (MCC) of the 'unseen' class vs. all other classes (i. e., the set of known classes). MCC is a popular measure for evaluating binary classification that accounts for the class imbalance (Chicco & Jurman, 2020). Dealing with this class imbalance is important as the number of vertices from the known classes is much larger than the number of vertices from the unseen class. It ranges from -1 to 1, where zero corresponds to a random prediction. In more detail, the MCC is computed as:

$$\text{MCC} = \frac{\text{TP} \cdot \text{TN} - \text{FP} \cdot \text{FN}}{\sqrt{(\text{TP}+\text{FP})(\text{TP}+\text{FN})(\text{TN}+\text{FP})(\text{TN}+\text{FN})}}$$

where TP are true positives or correctly rejected instances, TN true negatives, FP false positives, and FN false negatives. We accumulate those numbers over the entire sequence of tasks.

9.4. Results

The results for DBLP-easy are shown in Table 5 and the results for DBLP-hard in Table 6. For both DBLP-easy and DBLP-hard, we observe that the MCC scores, which measure the correct detection of new classes, are consistently higher for gDOC than for plain DOC. The same holds for the Open F1 Macro scores, which measure the overall performance of OOD detection + classification: gDOC is consistently better than plain DOC.

When comparing DBLP-easy and DBLP-hard, we see that the absolute F1 score and the MCC score attained on DBLP-easy are higher than the absolute scores on DBLP-hard, which is expected since DBLP-hard has more classes and DBLP-easy is the subset of data on which hyperparameters were tuned.

On DBLP-hard, the F1 score of plain DOC with a limited history size is very low: between 0.01 for history size 1 and 0.12 for

Table 5
Results for unseen class detection on **DBLP-easy** with GraphSAGE as base model (average of 5 repetitions). α indicates that risk reduction is used with the respective factor for the standard deviation, τ is the minimum threshold. Runs named gDOC are trained with weighted cross entropy. DOC is our baseline.

c	Open Learning Method	MCC		Open F1 Macro	
		cold	warm	cold	warm
1	DOC ($\tau = 0.50$)	.05	.07	.25	.25
	DOC ($\tau = 0.50, \alpha = 3.0$)	.05	.07	.25	.25
	gDOC ($\tau = 0.50$)	.04	**.08**	.30	**.33**
	gDOC ($\tau = 0.50, \alpha = 3.0$)	.04	.05	.30	.32
	gDOC ($\tau = 0.75$)	.04	.07	.30	.30
3	DOC ($\tau = 0.50$)	.05	.05	.28	.30
	DOC ($\tau = 0.50, \alpha = 3.0$)	.05	.05	.28	.30
	gDOC ($\tau = 0.50$)	.05	.08	.34	.34
	gDOC ($\tau = 0.50, \alpha = 3.0$)	.06	.08	.34	.34
	gDOC ($\tau = 0.75$)	.07	**.09**	.34	.34
6	DOC ($\tau = 0.50$)	.06	.06	.31	.32
	DOC ($\tau = 0.50, \alpha = 3.0$)	.06	.06	.31	.32
	gDOC ($\tau = 0.50$)	.07	.07	.35	.35
	gDOC ($\tau = 0.50, \alpha = 3.0$)	.07	.07	.35	.35
	gDOC ($\tau = 0.75$)	.09	**.10**	.35	.35
full	DOC ($\tau = 0.50$)	.07	.07	.32	.33
	DOC ($\tau = 0.50, \alpha = 3.0$)	.07	.07	.32	.33
	gDOC ($\tau = 0.50$)	.06	.06	.35	.35
	gDOC ($\tau = 0.50, \alpha = 3.0$)	.06	.06	.35	.35
	gDOC ($\tau = 0.75$)	.08	**.10**	.35	.35

Table 6
Results for unseen class detection on **DBLP-hard** with GraphSAGE as base model (average of 5 repetitions). α indicates that risk reduction is used with the respective factor for the standard deviation, τ is the minimum threshold. Runs named gDOC are trained with weighted cross entropy. DOC is our baseline.

c	Open Learning Method	MCC		Open F1 Macro	
		cold	warm	cold	warm
1	DOC ($\tau = 0.50$)	.01	.04	.01	.01
	DOC ($\tau = 0.50, \alpha = 3.0$)	.01	.02	.01	.01
	gDOC ($\tau = 0.50$)	.04	.05	**.13**	**.13**
	gDOC ($\tau = 0.50, \alpha = 3.0$)	.04	.05	**.13**	**.13**
	gDOC ($\tau = 0.75$)	.04	**.09**	**.13**	**.13**
3	DOC ($\tau = 0.50$)	.02	.03	.02	.05
	DOC ($\tau = 0.50, \alpha = 3.0$)	.02	.03	.02	.05
	gDOC ($\tau = 0.50$)	.05	.06	**.15**	**.15**
	gDOC ($\tau = 0.50, \alpha = 3.0$)	.05	.06	**.15**	**.15**
	gDOC ($\tau = 0.75$)	.05	**.08**	**.15**	**.15**
6	DOC ($\tau = 0.50$)	.02	.03	.05	.08
	DOC ($\tau = 0.50, \alpha = 3.0$)	.02	.03	.05	.08
	gDOC ($\tau = 0.50$)	.05	.06	**.16**	**.16**
	gDOC ($\tau = 0.50, \alpha = 3.0$)	.05	.06	**.16**	**.16**
	gDOC ($\tau = 0.75$)	.05	**.07**	**.16**	**.16**
full	DOC ($\tau = 0.50$)	.02	.04	.08	.12
	DOC ($\tau = 0.50, \alpha = 3.0$)	.02	.04	.08	.12
	gDOC ($\tau = 0.50$)	.04	.05	**.16**	**.16**
	gDOC ($\tau = 0.50, \alpha = 3.0$)	.05	.05	**.16**	**.16**
	gDOC ($\tau = 0.75$)	.05	**.07**	**.16**	**.16**

unlimited history size. In the same setting, gDOC achieves much higher scores: already 0.13 with a history size of 1 and 0.16 with at least a history size of 6. This shows that the class-weighted binary cross-entropy in gDOC is necessary to achieve reasonable F1 scores.

For thresholds, the results indicate that a high threshold (0.75) is preferable to lower thresholds. We further note that the combination of warm restarts and a small history size leads to the highest MCC score (0.09) on DBLP-hard, while on DBLP-easy, on which the hyperparameters have been tuned, the MCC score is higher for larger history sizes.

In Fig. 11, we show that risk reduction, i. e., lowering the detection threshold based on the class-specific standard deviation, does not help to increase performance. With a low minimum threshold (e. g.,0), we see the pure performance of the risk reduction technique, which peaks at $\alpha = 1$ before it decreases. When using a high minimum threshold (0.5, 0.75, 1.0), applying risk reduction only decreases the OOD performance. In other words, the absolute best OOD detection performance is achieved when the minimum threshold τ is set to 0.75, regardless of the risk reduction factor α. Therefore, the usefulness of risk reduction for our heavily imbalanced datasets is questionable.

To understand this result, we recall that risk reduction is a technique for calculating class-specific thresholds τ_i (see Section 3.2). However, this is only possible up to the global minimum threshold τ. Thus, even with risk reduction, class-specific thresholds cannot go below τ.

Table 7
Comparison of gDOC combined with different base models on DBLP-hard. The gDOC threshold is set to the 0.75 and no risk reduction is applied.

c	Method	ID accuracy		OOD MCC		Open F1	
		cold	warm	cold	warm	cold	warm
1	GS+gDOC	**36.4**	37.6	.04	.09	**.13**	.13
	SGC+gDOC	35.0	**38.4**	**.05**	**.10**	.12	**.14**
	GAT+gDOC	34.0	38.0	.04	.08	.10	.13
3	GS+gDOC	40.9	40.7	**.05**	**.08**	**.15**	.15
	SGC+gDOC	**41.6**	**41.9**	.05	.07	**.15**	**.16**
	GAT+gDOC	40.3	40.3	.04	.07	.13	.13
6	GS+gDOC	42.5	42.2	**.05**	**.07**	**.16**	**.16**
	SGC+gDOC	**43.7**	**43.4**	.04	**.07**	**.16**	**.16**
	GAT+gDOC	**43.7**	**43.4**	.04	**.07**	**.16**	**.16**
full	GS+gDOC	43.6	**43.5**	**.05**	**.07**	**.16**	**.16**
	SGC+gDOC	*out of GPU memory*					
	GAT+gDOC	**43.9**	**43.5**	.04	.05	**.16**	**.16**

9.5. Combining gDOC with different GNN base models

The gDOC module can be used in conjunction with arbitrary GNN base models. We compare GraphSAGE, GAT, and SGC as a base model for gDOC. We chose SGC because of its strong performance on the DBLP-hard dataset in Experiment 2, along with GAT as the most popular anisotropic model and GraphSAGE since it is one of the most popular isotropic models. As in the previous experiments, the learning rate was tuned for ID classification on DBLP-easy. Each configuration of history size and cold/warm restarts is independently optimized with respect to the hyperparameters.

The results are shown in Table 7. The ranking of the base models is similar to the results obtained in Experiment 2, which shows that adding the gDOC module has no unexpected effects on the base models. In particular, using SGC leads to similar performance as GraphSAGE. However, SGC exceeds 30 GB GPU memory on the full-history configuration and runs out of memory. GAT performs below GraphSAGE and SGC under smaller history size conditions, while it tends to catch up in terms of in-distribution accuracy and Macro-F1 when more history is available. In conclusion, this comparison confirms that gDOC can be successfully combined with various GNN base models.

9.6. Trade-off between in-distribution accuracy and OOD detection

We assess how in-distribution accuracy is affected by new class detection capabilities. Therefore, we report the average accuracy across tasks, calculated in the same way as in Experiment 2 from Section 7. The results are reported in Table 8 and show that, as expected, a plain GraphSAGE without OOD capabilities has a higher in-distribution accuracy than training with OOD detection capabilities (GraphSAGE+gDOC). This difference is caused by training with binary cross-entropy instead of the standard

Table 8
Trade-off between in-distribution classification accuracy and out-of-distribution detection performance on DBLP-hard. GraphSAGE (without an OOD detection module) is trained with categorical cross-entropy, while the methods capable of OOD detection are trained with binary cross-entropy. For ID accuracy, we always select the class with the maximum logit, regardless of any OOD threshold. NA marks no OOD detection capabilities.

c	Method	ID accuracy		OOD MCC	
		cold	warm	cold	warm
1	GraphSAGE+gDOC($\tau = 0.75$)	**36.4**	37.6	**.04**	**.09**
	GraphSAGE+DOC($\tau = 0.5, \alpha = 3.0$)	35.2	28.7	.01	.02
	GraphSAGE	34.5	**40.0**	NA	NA
3	GraphSAGE+gDOC($\tau = 0.75$)	40.9	40.7	**.05**	**.08**
	GraphSAGE+DOC($\tau = 0.5, \alpha = 3.0$)	39.4	43.1	.02	.03
	GraphSAGE	**44.3**	**45.1**	NA	NA
6	GraphSAGE+gDOC($\tau = 0.75$)	42.5	42.2	**.05**	**.07**
	GraphSAGE+DOC($\tau = 0.5, \alpha = 3.0$)	43.6	44.1	.02	.03
	GraphSAGE	**46.5**	**46.7**	NA	NA
full	GraphSAGE+gDOC($\tau = 0.75$)	43.6	43.5	**.05**	**.07**
	GraphSAGE+DOC($\tau = 0.5, \alpha = 3.0$)	42.9	45.1	.02	.04
	GraphSAGE	**46.8**	**47.1**	NA	NA

categorical cross-entropy. An interesting exception is that GraphSAGE+gDOC is better than GraphSAGE on the smallest history size configuration ($c = 1$). We assume that this difference is caused by GraphSAGE overfitting to the little data from a single graph snapshot, whereas the weighted cross-entropy of gDOC seems to alleviate this problem.

9.7. Summary

Our experiments have shown that weighting the binary cross-entropy loss function in gDOC is essential for unseen class detection in imbalanced graph data. We also learned that the risk reduction technique (as proposed in DOC (Shu et al., 2017)) is not helpful on our imbalanced graph datasets. That is because the variance among predictions in the unbalanced case is so high that the (minimum) threshold effectively never changed. The only exceptions are tiny factors of standard deviation (<1). Nevertheless, this only decreases the unseen class detection performance measured by MCC. We recommend using gDOC with weighted binary cross-entropy to account for class imbalance. However, we could not find any benefits of the risk reduction technique proposed in the original DOC. We have successfully combined gDOC with different base models (see Section 9.5) and analyzed the trade-off between ID accuracy and OOD detection capabilities (see Section 9.6).

10. General discussion

10.1. Main findings

Our experiments show several key results. First, we have shown in Section 6 that it is *not* necessary to up-train GNNs when new unlabeled data arrives. Instead, the performance of inductively pre-trained GNNs remains stable, even when new unlabeled data are added to the graph.

From the incremental training experiments with limited history sizes in Section 7, we obtain results that are almost as good as when using the entire history of the graph: With window sizes of 3 or 4 (50% receptive field coverage), GNNs achieve at least 95% accuracy compared to using all past data for incremental training. With window sizes of 6 or 8 (75% receptive field coverage), the GNN retains at least 99% accuracy. This result holds for standard GNN architectures and scalable and sampling-based approaches. This result directly impacts lifelong learning of GNNs in evolving graphs, as the setting closely resembles real-world applications. We have investigated whether to reuse parameters from previous tasks (warm restarts). We find that reusing an "old" model is a viable strategy, even though new classes appear during the sequence of tasks and the history size is limited. We have shown that reusing parameters from previous tasks becomes critical when the history sizes are small because less explicit knowledge is available.

We have shown in Section 8 that the methods work well, even when the labeled data are limited, which is essential for real-world applications because data annotation is expensive.

With the introduction of gDOC, we have made the first step to introduce new class detection in lifelong graph learning in Section 9, by combining graph neural networks with the DOC (Shu et al., 2017) module and extending it to take into account class imbalance. Our experiments on new class detection show that it is necessary to adjust the weights of binary cross-entropy training in gDOC to account for the imbalanced label distribution. Contrary to the original DOC, we have not observed any improvements with risk reduction through the standard deviation of logits. Instead, the best results were achieved with an appropriate threshold ($\tau = 0.75$) regardless of the risk reduction factor α. We acknowledge that emitting a crisp decision in unsupervised unseen class detection is a highly challenging problem.

Another interesting result is that combining warm restarts with small history sizes has increased MCC scores on the most challenging DBLP-hard dataset. It seems that omitting old data helps to detect out-of-distribution examples better.

10.2. Generalizability

We have shown that our incremental training approach can be applied to various GNN models and is orthogonal to sampling and preprocessing approaches. Our incremental training procedure can generally be applied to any GNN architecture with few caveats. If the GNN architecture depends on transductive learning, this constraint carries over to incremental training. Similarly, any pre-computation steps, such as computing normalizing constants such as in GCN (Kipf & Welling, 2017) or GraphSAINT (Zeng et al., 2020), must be performed again when adapting the model to a new task.

We assume in this work that old data do not change, e. g.,the vertices' labels remain the same over time. This is a reasonable assumption for citation and collaboration graphs. However, changes to old data may happen when generalizing the framework to other domains. This is addressed by the framework as follows: We use a certain history of the data for training defined by the history size c. Any change on older vertices like a label being changed would be immediately reflected in the following training iterations, i. e., the next tasks. In the cold start setting, the next trained model would be immediately be trained on the new correct ground truth data (up to history size). In the warm restart setting, the old parameters could still encode the impure knowledge. But ultimately it would also receive the up-to-date ground truth label as the training data for the next tasks.

To reflect our work in the broader context of lifelong or continual learning, we reconsider the gradient episodic memory framework (Lopez-Paz & Ranzato, 2017) for image data, in which the examples are independent. Specific pre-processing steps are required to cast graph data into independent examples for vertex classification, such as transforming each vertex into a graph (Wang et al., 2022). This increases the number of inference steps by $\mathcal{O}(|V|)$ compared to our approach.

11. Conclusion and future work

We have conducted extensive experiments to investigate how graph neural networks behave in a lifelong learning setting on evolving graph data in which the class distribution is highly imbalanced and the models need to adapt to new classes over time. In the first experiment, we have shown that it is not necessary to up-train GNNs on new unlabeled data. Based on this result, we have explored in a second experiment the case of an evolving graph in which new labeled vertices are continually added, including new classes over a sequence of tasks. The results show that parameter reuse allows us to retain a high level of accuracy, even with a limited history size. In the third experiment, we continued in this setup and tested the sensitivity to the label rate in the evolving graph setup, where we confirmed our previous finding. Lastly, in the fourth experiment, we compared our newly proposed gDOC extension against the simple adaption of DOC to graphs, showing that taking the class imbalance into account during training is crucial. We have shown that gDOC can be successfully combined with different GNN models. To facilitate our analyses, we have shown that the tdiff_k measure to derive the history sizes is equivariant to different temporal granularities. The measure tdiff_k quantifies the temporal differences along the edges in a temporal graph and is suitable to be reused independently from the other methods presented in this work. These results show a rich picture covering numerous challenges of applying graph neural networks in practical settings without retraining the model from scratch as soon as new data arrive.

As future work, we intend to explore and adapt more out-of-distribution approaches to graphs, e. g.,by using the IsoMax loss function (Macêdo & Ludermir, 2021). Another promising direction of future work would adapt ideas from the L2AC framework to graphs, i. e.,integrating explicit retrieval and similarity components. For the scope of this work, we have limited ourselves to techniques that provide a crisp decision rather than an OOD score because an OOD score requires validation data to tune the thresholds. Instead, the crisp unseen class detection methods presented here will apply directly to real-world applications. Next, it will be interesting to analyze why omitting old training data helps detect out-of-distribution examples. Although we have removed old data solely based on the vertex's time, future work might want to analyze different approaches to determine which vertices to keep and which to remove, given a limited "memory" budget. For example, keeping vertices with a high degree or page rank could be beneficial. Another direction of future work would be to explore when it is safe to actively shrink the output layer of the GNNs, e. g.,by looking at the final layer's weights. We envision that the results of this work will spur the development of new specialized techniques for lifelong open-world learning in evolving graphs.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Code is available at https://github.com/lgalke/lifelong-learning. Datasets are available at https://doi.org/10.5281/zenodo.3764770.

References

Aggarwal, C., & Subbian, K. (2014). Evolutionary network analysis: A survey. *ACM Computing Surveys*, *47*(1), http://dx.doi.org/10.1145/2601412.

Aurelio, Y. S., de Almeida, G. M., Castro, C. L., & de Pádua Braga, A. (2019). Learning from imbalanced data sets with weighted cross-entropy function. *Neural Processing Letters*, *50*(2), 1937–1949. http://dx.doi.org/10.1007/s11063-018-09977-1.

Bendale, A., & Boult, T. E. (2016). Towards open set deep networks. In *CVPR* (pp. 1563–1572). IEEE, http://dx.doi.org/10.1109/CVPR.2016.173.

Bojchevski, A., Klicpera, J., Perozzi, B., Kapoor, A., Blais, M., Rózemberczki, B., et al. (2020). Scaling graph neural networks with approximate PageRank. In *KDD* (pp. 2464–2473). ACM.

Bresson, X., & Laurent, T. (2017). Residual gated graph ConvNets. arXiv:1711.07553.

Cai, J., Wang, X., Guan, C., Tang, Y., Xu, J., Zhong, B., et al. (2022). Multimodal continual graph learning with neural architecture search. In *WWW* (pp. 1292–1300). ACM, http://dx.doi.org/10.1145/3485447.3512176.

Chen, Z., & Liu, B. (2018). *Synthesis lectures on artificial intelligence and machine learning*, *Lifelong machine learning* (2nd ed.). Morgan & Claypool Publishers, http://dx.doi.org/10.2200/S00832ED1V01Y201802AIM037.

Chen, J., Ma, T., & Xiao, C. (2018). FastGCN: Fast learning with graph convolutional networks via importance sampling. In *ICLR*. OpenReview.net.

Chen, X., Wang, J., & Xie, K. (2021). TrafficStream: A streaming traffic flow forecasting framework based on graph neural networks and continual learning. arXiv:2106.06273.

Chen, J., Zhu, J., & Song, L. (2018). Stochastic training of graph convolutional networks with variance reduction. In *ICML*.

Chiang, W., Liu, X., Si, S., Li, Y., Bengio, S., & Hsieh, C. (2019). Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks. In *KDD* (pp. 257–266). ACM.

Chicco, D., & Jurman, G. (2020). The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. *BMC Genomics*, *21*(1), 1–13.

Da, X., Chuanwei, R., Evren, K., Sushant, K., & Kannan, A. (2020). Inductive representation learning on temporal graphs. In *ICLR*. OpenReview.net.

Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2019). BERT: pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*. Association for Computational Linguistics.

Dhamija, A. R., Günther, M., & Boult, T. E. (2018). Reducing network agnostophobia. In *NeurIPS* (pp. 9175–9186).

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., et al. (2021). An image is worth 16 × 16 words: Transformers for image recognition at scale. In *ICLR*. OpenReview.net.

Dwivedi, V. P., Joshi, C. K., Laurent, T., Bengio, Y., & Bresson, X. (2020). Benchmarking graph neural networks. arXiv:2003.00982.

Febrinanto, F. G., Xia, F., Moore, K., Thapa, C., & Aggarwal, C. (2022). Graph lifelong learning: A survey. arXiv:2202.10688.

Fei, G., Wang, S., & Liu, B. (2016). Learning cumulatively to become more knowledgeable. In *KDD* (pp. 1565–1574). ACM.

Fey, M., & Lenssen, J. E. (2019). Fast graph representation learning with PyTorch Geometric. In *ICLR workshop on representation learning on graphs and manifolds*.

French, R. M. (1999). Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, *3*(4), 128–135. http://dx.doi.org/10.1016/S1364-6613(99)01294-2.

Galke, L., Franke, B., Zielke, T., & Scherp, A. (2021). Lifelong learning of graph neural networks for open-world node classification. In *IJCNN*. IEEE, http://dx.doi.org/10.1109/IJCNN52387.2021.9533412.

Galke, L., Mai, F., Vagliano, I., & Scherp, A. (2018). Multi-modal adversarial autoencoders for recommendations of citations and subject labels. In *UMAP*. ACM, http://dx.doi.org/10.1145/3209219.3209236.

Galke, L., Vagliano, I., & Scherp, A. (2019). Can graph neural networks go "online"? An analysis of pretraining and inference. In *Representation learning on graphs and manifolds, ICLR workshop*.

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*. JMLR.org.

Goodfellow, I. J., Bengio, Y., & Courville, A. C. (2016). *Adaptive computation and machine learning*, *Deep learning*. MIT Press.

Goyal, P., Chhetri, S. R., & Canedo, A. (2020). dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. *Knowledge-Based Systems*, *187*, http://dx.doi.org/10.1016/j.knosys.2019.06.024.

Goyal, P., Kamra, N., He, X., & Liu, Y. (2018). DynGEM: Deep embedding method for dynamic graphs. arXiv:1805.11273.

Hamilton, W. L. (2020). *Synthesis lectures on artificial intelligence and machine learning*, *Graph representation learning*. Morgan & Claypool Publishers, http://dx.doi.org/10.2200/S01045ED1V01Y202009AIM046.

Hamilton, W. L., Ying, Z., & Leskovec, J. (2017). Inductive representation learning on large graphs. In *NeurIPS*.

He, X., Deng, K., Wang, X., Li, Y., Zhang, Y., & Wang, M. (2020). LightGCN: Simplifying and powering graph convolution network for recommendation. In *SIGIR* (pp. 639–648). ACM, http://dx.doi.org/10.1145/3397271.3401063.
Hendrycks, D., Mazeika, M., & Dietterich, T. G. (2019). Deep anomaly detection with outlier exposure. In *ICLR*. OpenReview.net.
Herbster, M., Pontil, M., & Wainer, L. (2005). Online learning over graphs. *119*, In *ICML* (pp. 305–312). ACM.
Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., et al. (2020). Open graph benchmark: Datasets for machine learning on graphs. In *NeurIPS*.
Hu, Y., You, H., Wang, Z., Wang, Z., Zhou, E., & Gao, Y. (2021). Graph-MLP: Node classification without message passing in graph. arXiv preprint arXiv:2106.04051.
Huang, W., Zhang, T., Rong, Y., & Huang, J. (2018). Adaptive sampling towards fast graph representation learning. In *NeurIPS*.
Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. In *ICLR*. OpenReview.net.
Kipf, T. N., & Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *ICLR*. OpenReview.net.
Kumar, S., Zhang, X., & Leskovec, J. (2018). Learning dynamic embeddings from temporal interactions. arXiv:1812.02289.
Lee, K., Lee, K., Lee, H., & Shin, J. (2018). A simple unified framework for detecting out-of-distribution samples and adversarial attacks. In *NeurIPS* (pp. 7167–7177).
Lee, J. B., Nguyen, G., Rossi, R. A., Ahmed, N. K., Koh, E., & Kim, S. (2021). Dynamic node embeddings from edge streams. *IEEE Transactions on Emerging Topics in Computational Intelligence*, *5*(6), 931–946. http://dx.doi.org/10.1109/TETCI.2020.3011432.
Liang, S., Li, Y., & Srikant, R. (2018). Enhancing the reliability of out-of-distribution image detection in neural networks. In *ICLR*. OpenReview.net.
Lin, J. (1991). Divergence measures based on the Shannon entropy. *IEEE Transactions on Information Theory*, *37*(1), http://dx.doi.org/10.1109/18.61115.
Liu, B. (2017). Lifelong machine learning: a paradigm for continuous learning. *Frontiers of Computer Science*, *11*(3), 359–361. http://dx.doi.org/10.1007/s11704-016-6903-6.
Liu, H., Yang, Y., & Wang, X. (2021). Overcoming catastrophic forgetting in graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence, vol. 35, no. 10* (pp. 8653–8661). http://dx.doi.org/10.1609/aaai.v35i10.17049.
Lopez-Paz, D., & Ranzato, M. (2017). Gradient episodic memory for continual learning. In *NIPS* (pp. 6467–6476).
Macêdo, D., & Ludermir, T. (2021). Improving entropic out-of-distribution detection using isometric distances and the minimum distance score. arXiv:2105.14399.
Macêdo, D., Ren, T. I., Zanchettin, C., Oliveira, A. L., & Ludermir, T. (2021). Entropic out-of-distribution detection. In *IJCNN*. IEEE.
Mai, F., Galke, L., & Scherp, A. (2018). Using deep learning for title-based semantic subject indexing to reach competitive performance to full-text. In *Proceedings of the 18th ACM/IEEE on joint conference on digital libraries*. ACM, http://dx.doi.org/10.1145/3197026.3197039.
Manessi, F., Rozza, A., & Manzo, M. (2020). Dynamic graph convolutional networks. *Pattern Recognition*, *97*.
Masud, M. M., Gao, J., Khan, L., Han, J., & Thuraisingham, B. M. (2011). Classification and novel class detection in concept-drifting data streams under time constraints. *IEEE Transactions on Knowledge and Data Engineering*, *23*(6), 859–874. http://dx.doi.org/10.1109/TKDE.2010.61.
Monti, F., Boscaini, D., Masci, J., Rodolà, E., Svoboda, J., & Bronstein, M. M. (2017). Geometric deep learning on graphs and manifolds using mixture model CNNs. In *CVPR*. IEEE.
Newman, M. E. (2005). Power laws, Pareto distributions and Zipf's law. *Contemporary Physics*, *46*(5).
Nguyen, G. H., Lee, J. B., Rossi, R. A., Ahmed, N. K., Koh, E., & Kim, S. (2018). Continuous-time dynamic network embeddings. In *WWW* (pp. 969–976). ACM.
Pang, G., Shen, C., Cao, L., & van den Hengel, A. (2021). Deep learning for anomaly detection: A review. *ACM Computing Surveys*, *54*(2), 38:1–38:38. http://dx.doi.org/10.1145/3439950.
Pareja, A., Domeniconi, G., Chen, J., Ma, T., Suzumura, T., Kanezashi, H., et al. (2020). Evolvegcn: Evolving graph convolutional networks for dynamic graphs. In *AAAI* (pp. 5363–5370). AAAI Press.
Parisi, G. I., Kemker, R., Part, J. L., Kanan, C., & Wermter, S. (2019). Continual lifelong learning with neural networks: A review. *Neural Networks*, *113*, 54–71. http://dx.doi.org/10.1016/j.neunet.2019.01.012.
Rebuffi, S.-A., Kolesnikov, A., Sperl, G., & Lampert, C. H. (2017). Icarl: Incremental classifier and representation learning. In *CVPR*.
Robins, A. V. (1995). Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science*, *7*(2), 123–146.
Rossi, E., Chamberlain, B., Frasca, F., Eynard, D., Monti, F., & Bronstein, M. M. (2020). Temporal graph networks for deep learning on dynamic graphs. arXiv:2006.10637.
Rossi, E., Frasca, F., Chamberlain, B., Eynard, D., Bronstein, M. M., & Monti, F. (2020). SIGN: scalable inception graph neural networks. arXiv:2004.11198.
Ruvolo, P., & Eaton, E. (2013). ELLA: an efficient lifelong learning algorithm. In *ICML* (pp. 507–515). JMLR.org.
Sankar, A., Wu, Y., Gou, L., Zhang, W., & Yang, H. (2020). DySAT: Deep neural representation learning on dynamic graphs via self-attention networks. In *WSDM*. ACM.
Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G. (2009). The graph neural network model. *IEEE Trans. Neural Networks*, *20*(1), http://dx.doi.org/10.1109/TNN.2008.2005605.
Sen, P., Namata, G., Bilgic, M., Getoor, L., Gallagher, B., & Eliassi-Rad, T. (2008). Collective classification in network data. *AI Magazine*, *29*(3).
Seo, Y., Defferrard, M., Vandergheynst, P., & Bresson, X. (2018). Structured sequence modeling with graph convolutional recurrent networks. In *ICONIP*. Springer.
Shu, L., Xu, H., & Liu, B. (2017). DOC: deep open classification of text documents. In *EMNLP* (pp. 2911–2916). ACL, http://dx.doi.org/10.18653/v1/d17-1314.
Silver, D. L., Yang, Q., & Li, L. (2013). Lifelong machine learning systems: Beyond learning algorithms. *SS-13-05*, In *AAAI spring symposium: Lifelong machine learning*. AAAI.
Tan, Z., Ding, K., Guo, R., & Liu, H. (2022). Graph few-shot class-incremental learning. In *WSDM* (pp. 987–996). ACM, http://dx.doi.org/10.1145/3488560.3498455.
Thrun, S. (1998). Lifelong learning algorithms. In *Learning to learn* (pp. 181–209). Springer.
Thrun, S., & Mitchell, T. M. (1995). Learning one more thing. In *IJCAI* (pp. 1217–1225). Morgan Kaufmann.
Trivedi, R., Dai, H., Wang, Y., & Song, L. (2017). Know-Evolve: Deep temporal reasoning for dynamic knowledge graphs. In *ICML*. PMLR.
Trivedi, R., Farajtabar, M., Biswal, P., & Zha, H. (2019). Dyrep: Learning representations over dynamic graphs. In *ICLR*. OpenReview.net.
Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2018). Graph attention networks. In *ICLR*. OpenReview.net.
Wang, B., Chen, Y., Li, X., & Chen, J. (2021). Lifelong classification in open world with limited storage requirements. *Neural Computation*, *33*(7), 1818–1852. http://dx.doi.org/10.1162/neco_a_01391.
Wang, C., Qiu, Y., Gao, D., & Scherer, S. (2022). Lifelong graph learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 13719–13728).
Wang, J., Song, G., Wu, Y., & Wang, L. (2020). Streaming graph neural networks via continual learning. In *CIKM* (pp. 1515–1524). ACM, http://dx.doi.org/10.1145/3340531.3411963.
Wang, M., et al. (2019). Deep graph library: Towards efficient and scalable deep learning on graphs. arXiv preprint arXiv:1909.01315.
Webb, G. I., Hyde, R., Cao, H., Nguyen, H. L., & Petitjean, F. (2016). Characterizing concept drift. *Data Mining and Knowledge Discovery*, *30*(4), 964–994.
Webb, G. I., Lee, L. K., Goethals, B., & Petitjean, F. (2018). Analyzing concept drift and shift from sample data. *Data Mining and Knowledge Discovery*, *32*(5), 1179–1199.
Wu, M., Pan, S., & Zhu, X. (2020). OpenWGL: Open-world graph learning. In *ICDM* (pp. 681–690). IEEE.
Wu, F., Souza, . A. H., Zhang, T., Fifty, C., Yu, T., & Weinberger, K. Q. (2019). Simplifying graph convolutional networks. In *ICML* (pp. 6861–6871). PMLR.
Xu, K., Hu, W., Leskovec, J., & Jegelka, S. (2019). How powerful are graph neural networks? In *ICLR*. OpenReview.net.
Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K., & Jegelka, S. (2018). Representation learning on graphs with jumping knowledge networks. In *ICML*. PMLR.
Xu, H., Liu, B., Shu, L., & Yu, P. (2019). Open-world Learning and Application to Product Classification. In *WWW* (pp. 3413–3419). ACM, http://dx.doi.org/10.1145/3308558.3313644.
Yang, Z., Cohen, W. W., & Salakhutdinov, R. (2016). Revisiting semi-supervised learning with graph embeddings. In *ICML*. JMLR.org.
Yang, J., Zhou, K., Li, Y., & Liu, Z. (2021). Generalized out-of-distribution detection: A survey. arXiv arXiv:2110.11334.
Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W. L., & Leskovec, J. (2018). Hierarchical graph representation learning with differentiable pooling. In *NeurIPS*.
Zeng, H., Zhou, H., Srivastava, A., Kannan, R., & Prasanna, V. K. (2020). Graphsaint: Graph sampling based inductive learning method. In *ICLR*. OpenReview.net.
Zhou, F., & Cao, C. (2021). Overcoming catastrophic forgetting in graph neural networks with experience replay. arXiv preprint arXiv:2003.09908.