BRIDGING GRAPH NETWORK TO LIFELONG LEARNING WITH FEATURE INTERACTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Graph neural networks (GNN) are powerful models for many graph-structured tasks. In this paper, we aim to bridge GNN to lifelong learning, which is to overcome the effect of "catastrophic forgetting" for continuously learning a sequence of graph-structured tasks. Although many lifelong learning techniques for convolutional neural networks (CNN) have been developed, lifelong learning for GNN is still underexplored and suffers from incomplete graph structure during learning. This is because in lifelong learning the nodes increase dynamically and can only be present to the model once, which makes many graph models and sampling strategies inapplicable. To solve this problem, we propose a new graph topology based on feature interaction, called the feature graph. It takes features as new nodes and turns nodes into independent graphs. This successfully converts the original problem of node classification to graph classification, in which the increasing nodes are turned into training samples. Therefore, the lifelong learning techniques developed for CNN become applicable to GNN for the first time. In the experiments, we demonstrate both the efficiency and effectiveness of feature graphs for lifelong learning tasks using a rehearsal method. We expect that it will have broad potential applications for graph-structured tasks in lifelong learning.

1 INTRODUCTION

Graph neural networks (GNN) have received increasing attention and proved useful for many tasks with graph-structured data, such as citation, social, and protein networks (Veličković et al., 2018). In this paper, we aim to extend graph neural networks to lifelong learning, which is also referred to as continual or incremental learning (Lopez-Paz & Ranzato, 2017). Concretely, lifelong learning aims at breaking the barrier between training and testing, and the goal is to develop algorithms that are able to keep updating the model parameters over time (Aljundi et al., 2019a). In other words, we expect the model to gradually accumulate knowledge based on a data (task) stream, where each sample can only be present to the model once. Intuitively, it is easy to suffer from the problem of "catastrophic forgetting" if the model is simply updated for new samples via gradient descent by the back-propagation algorithm (Rumelhart et al., 1986).

Although some strategies have been developed to alleviate the forgetting problem for convolutional neural networks (CNN), lifelong learning is still very difficult for graph networks. This is because most of existing graph models such as graph convolutional networks (GCN) (Kipf & Welling, 2017) require the entire graph for training. This is infeasible for lifelong learning, as the graph size can increase over time and we have to drop old data to learn new knowledge. To reduce the memory consumption, some graph sampling strategies were proposed, but they are also difficult to directly apply to lifelong learning (Hamilton et al., 2017; Chen et al., 2018b; Zeng et al., 2020). Moreover, since the training samples are not identically and independently distributed (*iid*) and only available in a continuum (Zenke et al., 2017), the sampling strategies that require pre-processing of the entire graph become infeasible (Zeng et al., 2020). As far as we know, lifelong learning for GNN is still an underexplored topic but has broad potential applications (Wu et al., 2020).

Recall that regular CNNs are trained in a mini-batch manner where the model can take samples as independent inputs (Krizhevsky et al., 2012). Our question is that can we convert GNN into a traditional CNN-like classification problem, so that (i) nodes can be predicted independently and (ii) the lifelong learning techniques developed for CNN can be easily adopted for GNN? However, this

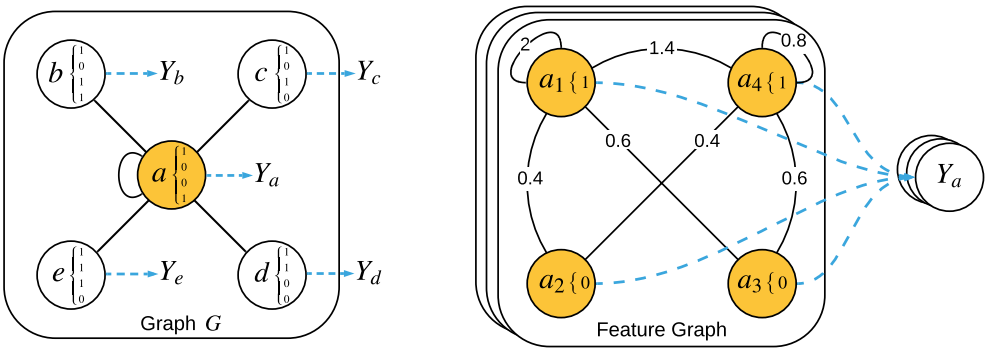


Figure 1: The regular graph networks learn a predictor for node classification, while the feature graphs take the features as nodes and turn nodes into graphs, resulting in a graph predictor, as each node in graph G has an associated feature graph. Take the node a for example, its features $\mathbf{x}_a = [1, 0, 0, 1]$ are nodes $\{a_1, a_2, a_3, a_4\}$ in feature graph. Its feature adjacency matrix is established via feature interaction between a and its neighbors $\mathcal{N}(a) = \{a, b, c, d, e\}$. This makes any lifelong learning techniques for CNN applicable to GNN, as nodes become training samples (feature graphs).

is not straightforward as node connections cannot be considered in a regular classification model. To solve this problem, we propose to construct a different graph topology, the feature graph in Figure 1, to bridge GNN to lifelong learning. It takes the features as nodes and turns the nodes into graphs. This converts the problem of node classification to graph classification. In this way, the increasing nodes become training samples, thus the feature graphs can be trained in a native mini-batch manner. Note that we do not focus on specific lifelong learning techniques or graph models, instead we try to overcome the difficulty of directly applying lifelong learning techniques to graph networks. We will demonstrate its effectiveness and efficiency using a rehearsal method (Aljundi et al., 2019b).

A feature graph does not retain the intuitive meaning of a regular graph. Take a social network for example, in a regular graph, each user is a node and user connections are edges. In feature graphs, each user is a graph, the user features are nodes, including age, gender, etc. Therefore, user classification become a graph classification problem and the increasing users become multiple training samples. Since the feature number is more stable than the node number in a regular graph, thus the size of a feature graph is more stable than its regular graph, which makes classification easier. We list their detailed relationship in Table 1 and will further explain them in Section 4.

Table 1: The relationship between a regular graph and its feature graph.

Graph		Feature Graph	Relationship		
Node	$\mapsto$	Graph	Node classification	$\mapsto$	Graph classification
Feature	$\mapsto$	Node	Fixed feature dimension	$\mapsto$	Fixed number of nodes
Edge	$\mapsto$	Edges	Adjacency matrix	$\mapsto$	Dynamic adjacency
Graph	$\mapsto$	Samples	One dynamic graph	$\mapsto$	Multiple graphs/samples

The most difficult and important step to create feature graphs is to establish the feature node connections (edges). We observe that the node features of a regular graph are often propagated in an element-wise manner (Kipf & Welling, 2017; Veličković et al., 2018; Klicpera et al., 2019; Zeng et al., 2020), which means that existing methods cannot model feature interactions well. To solve this problem, we propose to define the feature adjacency matrix as features' cross-correlation with neighbor nodes' features, which allows the feature graph to learn feature interactions explicitly. This is crucial for many graph-structured tasks, as useful information might be encoded in one's neighbor features. Moreover, feature graph is only of constant computational complexity with the increased learning tasks. We will demonstrate both the effectiveness and efficiency of the feature graphs in the experiments. To the best of our knowledge, it might the first practical method, which overcomes the difficulty of directly applying lifelong learning techniques to graph networks.

2 RELATED WORK

The existing lifelong graph learning methods are very limited, thus we will present the related work on lifelong learning and graph neural networks, respectively.

2.1 LIFELONG LEARNING

Non-rehearsal methods Methods in the first category do not preserve any old data. To alleviate the forgetting problem, progressive neural networks (Rusu et al., 2016) leveraged prior knowledge via lateral connections to previously learned features. Learning with forgetting (LwF) (Li & Hoiem, 2017) introduced a knowledge distillation loss (Hinton et al., 2015) to neural networks, which encouraged the network output for new classes to be close to the original outputs. Distillation loss was also applied to learning object detectors incrementally (Shmelkov et al., 2017). Learning without memorizing (LwM) (Dhar et al., 2019) extended LwF by adding an attention distillation term based on attention maps and demonstrated that it is helpful for retaining information of the old classes. EWC (Kirkpatrick et al., 2017) remembered old tasks by slowing down learning on important weights. RWalk (Chaudhry et al., 2018) generalized EWC and improved weight consolidation by adding a KL-divergence-based regularization. Memory aware synapses (MAS) (Aljundi et al., 2018) computed an importance for each parameter in an unsupervised manner based on the sensitivity of output function to parameter changes. A sparse writing protocol is introduced to a memory module (Wang et al., 2020), ensuring that only a few memory spaces can be affected during training.

Rehearsal methods Rehearsal methods can be roughly divided into rehearsal with synthetic data or exemplars from old data (Robins, 1995). For example, to ensure that the loss of exemplars does not increase, gradient episodic memory (GEM) (Lopez-Paz & Ranzato, 2017) introduced orientation constraints during gradient updates. Inspired by GEM, Aljundi et al. (2019b) selected exemplars with a maximal cosine similarity of the gradient orientation. iCaRL (Rebuffi et al., 2017) preserved a subset of images with a herding algorithm (Welling, 2009) and included the subset when updating the network for new classes. EEIL (Castro et al., 2018) extended iCaRL by learning the classifier in an end-to-end manner. Wu et al. (2019) further extended iCaRL and discovered that updating the model with class-balanced exemplars further improved the performance. Similarly, Hou et al. (2019); Belouadah & Popescu (2019) further added constraints to the loss function to mitigate the effect of imbalance. To reduce the memory consumption of exemplars, Iscen et al. (2020) applied the distillation loss to feature space without having to access to the corresponding images. Several exemplar selection methods are tested in (Zhou et al., 2020) for graph networks, while it has a different problem formulation with this paper. Rehearsal approaches with synthetic data based on generative adversary networks (GAN) were used to reduce the dependence of the exemplars on old data (Shin et al., 2017; Wu et al., 2018; He et al., 2018; Xiang et al., 2019).

2.2 GRAPH NEURAL NETWORKS

Graph neural networks have been widely used to solve problems about graph-structured data (Zhou et al., 2018). The spectral network was the first to extend convolution to the graph domain (Bruna et al., 2014). Later the graph convolutional network (GCN) (Kipf & Welling, 2017) followed this idea and alleviated the problem of overfitting on local neighborhoods via first order Chebyshev expansion. This obtained a good generalization ability and inspired more recent work. For example, to identify the importance of neighborhood features, graph attention network (GAT) (Veličković et al., 2018) added an attention mechanism into GCN, which further improved the performance on citation networks and the protein-protein interaction dataset.

However, GCN and its variants require the entire graph during training, thus they cannot scale to large graphs. To solve this problem and train GNN with mini-batches, many layer sampling techniques have been introduced. For example, GraphSAGE (Hamilton et al., 2017) learned a function to generate node embedding by sampling and aggregating neighborhood features. JK-Net (Xu et al., 2018) followed the same sampling strategy and demonstrated a significant accuracy improvement on GCN with more than two layers. DiffPool (Ying et al., 2018b) learned a differentiable soft cluster assignment to map nodes to a set of clusters, which then formed a coarsened input for the next layer. Ying et al. (2018a) designed a training strategy that relied on harder-and-harder training examples to improve the robustness and convergence speed of the model. S-GCN (Chen et al., 2018a) devel-

oped control variate-based algorithms to sample small neighbor sizes and used historical activation in the previous layer to avoid redundant re-evaluation. FastGCN (Chen et al., 2018b) applied importance sampling to reduce variance and performed node sampling for each layer independently, which resulted in a constant sample size in all layers. Huang et al. (2018) sampled the lower layer conditioned on the top one ensuring higher accuracy and fixed-size sampling. Subgraph sampling techniques were also developed to reduce memory consumption. ClusterGCN (Chiang et al., 2019) sampled a block of nodes in a dense subgraph identified by a clustering algorithm and restricted the neighborhood search within the subgraph. GraphSiant (Zeng et al., 2020) constructed mini-batches by sampling the training graph. A complete GCN with a fixed number of well-connected nodes is built from the properly sampled subgraph during each iteration.

Nevertheless, most of the sampling techniques still require a pre-processing of the entire graph to determine the graph sampling process or require a complete graph structure, which makes those algorithms not directly applicable to lifelong learning. In this paper, we hypothesize that a different graph structure is required for lifelong learning, and the new structure is not necessary to maintain the original/intuitive meaning of the nodes and edges.

3 Problem Formulation

We start from defining regular graph learning before lifelong graph learning for coherence. An attribute graph is defined as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a set of nodes and $\mathcal{E} \subseteq \left\{\{v_a, v_b\} \,|(v_a, v_b) \in \mathcal{V}^2\right\}$ is a set of edges. Each node $v \in \mathcal{V}$ is associated with a target vector $\mathbf{z}_v \in \mathcal{Z}$ and a multi-channel feature vector $\mathbf{x}_v \in \mathcal{X} \subset \mathbb{R}^{F \times C}$. Each edge $e \in \mathcal{E}$ is associated with a weight vector $\mathbf{w}_e \in \mathcal{W} \subset \mathbb{R}^W$. In regular graph learning (Kipf & Welling, 2017), we aim to learn a predictor f to predict the target vector $\mathbf{z}_v$ associated with a node $\mathbf{x}_v, v \in \mathcal{V}'$, given the graph $\mathcal{G}$, nodes features $\mathcal{X}$, weights vectors $\mathcal{W}$, and part of the targets $\mathbf{z}_v, v \in \mathcal{V} \setminus \mathcal{V}'$.

In lifelong graph learning, we have the same objective with the regular graph learning, but we are only able to obtain the graph-structured data as a data continuum $\mathcal{G}_L = \{(\mathbf{x}_i, t_i, \mathbf{z}_i, \mathcal{N}_{k=1:K}(\mathbf{x}_i), \mathcal{W}_{k=1:K}(\mathbf{x}_i))_{i=1:N}\}$, where each item is formed by a node feature $\mathbf{x}_i \in \mathcal{X}$, a task descriptor $t_i \in \mathcal{T}$, a target vector $\mathbf{z}_i \in \mathcal{Z}_{t_i}$, a k-hop neighbor set $\mathcal{N}_{k=1:K}(\mathbf{x}_i)$, and a weight vector set $\mathcal{W}_{k=1:K}(\mathbf{x}_i)$ associated with the k-hop neighbors. For simplicity, we will use the symbol $\mathcal{N}(\mathbf{x}_i)$ to denote the available neighbor set and their associated weights. We assume that every item $(\mathbf{x}_i, \mathcal{N}(\mathbf{x}_i), t_i, \mathbf{z}_i)$ satisfies $(\mathbf{x}_i, \mathcal{N}(\mathbf{x}_i), \mathbf{z}_i) \sim P_{t_i}(\mathcal{X}, \mathcal{N}(\mathcal{X}), \mathcal{Z})$, where P_{t_i} is a probability distribution and describes a single learning task. In lifelong graph learning, we will observe, example by example, the continuum of the graph-structured data as

$$(\mathbf{x}_1, \mathcal{N}(\mathbf{x}_1), t_1, \mathbf{z}_1), \ldots, (\mathbf{x}_i, \mathcal{N}(\mathbf{x}_i), t_i, \mathbf{z}_i), \ldots, (\mathbf{x}_N, \mathcal{N}(\mathbf{x}_N), t_N, \mathbf{z}_N). \tag{1}$$

While observing (1), our goal is to learn a predictor f_L to associate a test sample $(\mathbf{x}, \mathcal{N}(\mathbf{x}), t)$ with a target $\mathbf{z}$ such that $(\mathbf{x}, \mathcal{N}(\mathbf{x}), \mathbf{z}) \sim P_t$. Such test sample can belong to a task that we have observed in the past, the current task, or a task that we will observe (or not) in the future.

Note that examples are **not** drawn locally identically and independently distributed (*iid*) from a fixed probability distribution, since we don't know the task boundary and a long sequence from the current task may be observed before switching to the next task. In the continuum (1), we only know the label of $\mathbf{x}_i$, but have no information about the labels of its neighbors $\mathcal{N}(\mathbf{x}_i)$. In the experiments, a maximum $k = 1$ is required and the task descriptors are integers $t_i \in \mathbb{Z}$ enumerating different tasks.

4 Feature Graph Network

To better show the relationship with a regular graph, we first review the graph convolutional network (GCN) (Kipf & Welling, 2017). Given an attribute graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ described in Section 3, the stacked node features can be written as $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N]^{\mathrm{T}} \in \mathbb{R}^{N \times FC}$, where $\mathbf{x}_i \in \mathbf{R}^{FC}$ is a vectorized node feature. The GCN takes feature channel as $C = 1$, so that $\mathbf{x}_i \in \mathbf{R}^F, \mathbf{X} \in \mathbb{R}^{N \times F}$. In this sense, the l-th graph convolutional layer in GCN can be defined as

$$\mathbf{X}_{(l+1)} = \sigma\left(\hat{\mathbf{A}} \cdot \mathbf{X}_{(l)} \cdot \mathbf{W}\right), \tag{2}$$

where $\sigma(\cdot)$ is a non-linear activation function, $\mathbf{W} \in \mathbb{R}^{F_{(l)} \times F_{(l+1)}}$ is a learnable parameter, and $\hat{\mathbf{A}} \in \mathbb{R}^{N \times N}$ is the symmetrically normalized adjacency matrix $\mathbf{A}$ with self-loops.

$$\hat{\mathbf{A}} = \mathbf{D}^{-1/2} \tilde{\mathbf{A}} \mathbf{D}^{-1/2}, \quad \mathbf{D}_{[i,i]} = \sum_k \tilde{\mathbf{A}}_{[i,k]}, \quad \tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_N. \tag{3}$$

For unweighted graph, $\mathbf{A}_{[i,j]} = 1$ for $(i,j) \in \mathcal{E}$ and $\mathbf{A}_{[i,j]} = 0$ otherwise. The node features $\mathbf{x}_{i=1:N}$ share the same learnable weights $\mathbf{W}$, thus it can be regarded as a specific convolution. It doesn't change the number of nodes (rows of $\mathbf{X}$), but is able to change the feature dimension $F_{(l)}$ to $F_{(l+1)}$.

GCN has been applied to many graph-structured tasks due to its simplicity and good generalization ability. However, the problems of GCN are also obvious. The node features in the next layer is a linear combination of the node features in the current layer, thus its graph propagation cannot directly model the feature interaction. Moreover, to propagate node features across layers, a full adjacency matrix $\mathbf{A}$ is required, which means that the entire graph topology has to present to the model. This limits its usage to large graphs and makes it inapplicable to lifelong learning either. It becomes more severe when the neighbor features are concatenated and propagated to the next layer iteratively (Xu et al., 2019), resulting in the effect of "neighbor explosion." As mentioned in Section 2, to scale GCN to large graphs, many sampling techniques are proposed, however, most of them are still difficult to directly apply to lifelong learning. Therefore, we argue that such difficulty is caused by the inappropriate graph topology and propose to establish the feature graphs.

4.1 Feature Nodes

Recall that each node v in a regular graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is associated with a multi-channel feature vector $\mathbf{x} = \left[\mathbf{x}_{[1,:]}^{\mathrm{T}}, \cdots, \mathbf{x}_{[F,:]}^{\mathrm{T}}\right] \in \mathbb{R}^{F \times C}$, where $\mathbf{x}_{[i,:]}$ is the i-th feature (row) of $\mathbf{x}$. An attribute feature graph takes the features of a regular graph as nodes. It can be defined as $\mathcal{G}^{\mathcal{F}} = (\mathcal{V}^{\mathcal{F}}, \mathcal{E}^{\mathcal{F}})$, where each node $v^{\mathcal{F}} \in \mathcal{V}^{\mathcal{F}}$ is associated with a feature $\mathbf{x}_{[i,:]}^{\mathrm{T}}$ and will be denoted as $\mathbf{x}_i^{\mathcal{F}}$. Intuitively, the number of nodes in the feature graph is the feature dimension in the regular graph, *i.e.*, $\left|\mathcal{V}^{\mathcal{F}}\right| = F$, and the feature dimension in feature graph is the feature channel in a regular graph, *i.e.*, $\mathbf{x}_i^{\mathcal{F}} \in \mathbb{R}^C$. Therefore, we can define the feature nodes for feature graph as

$$\mathcal{V}^{\mathcal{F}} = \left\{\mathbf{x}_1^{\mathcal{F}}, \mathbf{x}_2^{\mathcal{F}}, \cdots, \mathbf{x}_i^{\mathcal{F}}, \cdots, \mathbf{x}_F^{\mathcal{F}}\right\}. \tag{4}$$

In this way, for each node $v \in \mathcal{V}$, we can construct an associated feature graph $\mathcal{G}^{\mathcal{F}}$. We next establish their associated feature adjacency matrices via feature interaction.

4.2 Feature Adjacency Matrix

For each item in continuum (1), the edges between $\mathbf{x}$ and its neighbors $\mathcal{N}(\mathbf{x})$ imply the relationship of their features. Therefore, feature interaction can be modeled as the cross-correlation matrix of the feature vectors, so that the k-hop c-th channel feature adjacency matrix can be defined as:

$$\mathbf{A}_{k,c}^{\mathcal{F}}(\mathbf{x}) \triangleq \mathrm{sgnroot}\left(\mathbb{E}\left[w_{x,y} \mathbf{x}_{[:,c]} \mathbf{y}_{[:,c]}^{\mathrm{T}}\right]\right), \quad \forall \mathbf{y} \in \mathcal{N}_k(\mathbf{x}), \quad c = 1, 2, \cdots, C, \tag{5}$$

where $\mathrm{sgnroot}(x) = \mathrm{sign}(x)\sqrt{(|x|)}$ and $w_{x,y}$ is the associated edge weight. Note that $\mathbf{x}_{[:,c]}$ in (5) is the c-th column (channel) of $\mathbf{x}$, while the feature node $\mathbf{x}_{[i,:]}$ is the i-th row of $\mathbf{x}$. Intuitively, we have $\mathbf{A}_{k,c}^{\mathcal{F}}(\mathbf{x}) \in \mathbb{R}^{F \times F}$, where $F \ll N$ due to the settings of lifelong learning. We can calculate multiple feature adjacency matrices if the edge weight is a high dimensional vector. In practice, the expectation in (5) is approximated by averaging over the observed neighbors.

$$\mathbb{E}\left[w_{x,y} \mathbf{x}_{[:,c]} \mathbf{y}_{[:,c]}^{\mathrm{T}}\right] \approx \frac{1}{|\mathcal{N}_k(\mathbf{x})|} \sum_{\mathbf{y} \in \mathcal{N}_k(\mathbf{x})} w_{x,y} \mathbf{x}_{[:,c]} \mathbf{y}_{[:,c]}^{\mathrm{T}}, \tag{6}$$

where we we need to change $\mathbf{x}_{[:,c]} \mathbf{y}_{[:,c]}^{\mathrm{T}}$ to $\left(\mathbf{x}_{[:,c]} \mathbf{y}_{[:,c]}^{\mathrm{T}} + \mathbf{y}_{[:,c]} \mathbf{x}_{[:,c]}^{\mathrm{T}}\right)$ for undirected graphs. Therefore, the feature adjacency matrix can be constructed online via (6), together with the availability of new samples in the continuum. In this way, the continuum (1) is converted to

$$(\mathcal{G}_1^{\mathcal{F}}, t_1, \mathbf{z}_1), \ldots, (\mathcal{G}_i^{\mathcal{F}}, t_i, \mathbf{z}_i), \ldots, (\mathcal{G}_N^{\mathcal{F}}, t_N, \mathbf{z}_N), \tag{7}$$

where $\mathcal{G}_i^{\mathcal{F}} = \left(\mathcal{V}^{\mathcal{F}}, \mathbf{A}^{\mathcal{F}}\right)$. This means that our objective becomes learning a graph predictor

$$f^{\mathcal{F}} : \mathcal{G}^{\mathcal{F}} \times \mathcal{T} \mapsto \mathcal{Z}, \tag{8}$$

to associate a test sample $(\mathcal{G}^{\mathcal{F}}, t)$ to target vector $\mathbf{z}$ such that $(\mathcal{G}^{\mathcal{F}}, \mathbf{z}) \sim P_t^{\mathcal{F}}$. Note that the samples in the new continuum (7) are still non-*iid*. In this way, a lifelong graph learning problem becomes a regular lifelong learning problem similar to (Lopez-Paz & Ranzato, 2017).

4.3 Feature Broadcast Layer

Given the feature graph $\mathcal{G}^{\mathcal{F}}$, the l-th feature broadcast layer can be defined as

$$\mathbf{x}_{(l+1)}^{\mathcal{F}} = \sigma\left(\hat{\mathbf{A}}_k^{\mathcal{F}} \cdot \mathbf{x}_{(l)}^{\mathcal{F}} \cdot \mathbf{W}^{\mathcal{F}}\right), \tag{9}$$

where $\sigma(\cdot)$ is a non-linear activation function, $\hat{\mathbf{A}}_k^{\mathcal{F}}$ is the associated normalized feature adjacency matrix, and $\mathbf{W} \in \mathbb{R}^{C_{(l)} \times C_{(l+1)}}$ is a learnable parameter. It is worth noting that, although the definition of (9) appears similar to (2), they have totally different meanings. The node feature $\mathbf{x}^{\mathcal{F}}$ in (9) can be a row of $\mathbf{X}$ in (2) if $C = 1$. In this sense, the increasing nodes in a regular graph becomes feature graphs with different adjacency matrices. Similar to (2), the feature broadcast layer doesn't change the number of feature nodes. However, this is not always necessary for feature graphs, as our objective has been turned into graph classification. Therefore, we also define a feature transform layer to change the number of feature nodes and present it in Appendix E due to limited pages. Note that we can also construct other types of layers based on the topology $\mathcal{G}^{\mathcal{F}}$, *e.g.*, a graph attention layer (Veličković et al., 2018) or extend convolution to kervolution (Wang et al., 2019). However, they are out of the scope of this paper, we will demonstrate the effectiveness of the proposed feature graph using the simple feature broadcast layer and feature transform layer in the experiments.

4.4 Assumption, Analysis, and Computational Complexity

As mentioned before, we assume that some useful information of a regular node is encoded in its neighbors' features. Therefore, we expect that our feature graph can perform better because of the explicit modeling of feature interaction. Intuitively, a typical regular graph is propagated by calculating $\hat{\mathbf{A}}\mathbf{X} = \hat{\mathbf{A}}\left[\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\right]^{\mathrm{T}}$, which means that the node features in the next layer are element-wise weighted average of the current node features. As a comparison, a feature of feature graphs are able to interact with another by calculating $\hat{\mathbf{A}}^{\mathcal{F}}\mathbf{x}^{\mathcal{F}}$, where the feature adjacency matrix $\mathbf{A}^{\mathcal{F}}$ is a function of cross-correlation matrix of features. This can be crucial for many graph-structured tasks that node features or part of the features have a strong relationship with each other.

Feature graphs have a low computational complexity. Intuitively, the complexity for calculating the feature adjacency matrix is $\mathcal{O}(nF_{(l)}^2)$, where n is the expected number of neighbors of each node in the continuum. Regardless of the number of layers which is a constant for specific models, the complexity of graph propagation for a feature broadcast layer and a feature transform layer is of $\mathcal{O}(F_{(l)}^2 C_{(l)} C_{(l+1)})$ and $\mathcal{O}(F_{(l)} F_{(l+1)} C_{(l)}^2)$, respectively. If we take $E_{(l)} = F_{(l)} C_{(l)}$ as the number of elements of the features, the complexity for calculating each new item in the continuum is roughly of $\mathcal{O}(E^2 + nF^2)$, where we leave out the layer index for simplicity. Note that they are irrelevant to the task number, thus feature graphs have constant complexity with the increased learning tasks.

4.5 Lifelong Learning

Since our objective has become a regular lifelong learning problem by defining the feature graph continuum (7), our feature graph models can inherit the achievements in lifelong learning developed for CNN, such as the weight consolidation (Kirkpatrick et al., 2017) or knowledge distillation (Rebuffi et al., 2017). This further verifies the generalization ability of feature graphs. As we only aim to demonstrate its effectiveness in lifelong learning, we will adopt a simple yet effective rehearsal method with random sample selection (Aljundi et al., 2019b). As reported in (Aljundi et al., 2019b), a uniform sampling tends to produce lower selection probability for earlier items. To compensate for such effect, we set the probability that the n-th observed item is selected at time t as

$$P_n(t) = \begin{cases} 1 & t \leqslant M \\ {}^{M}/_{n} & t > M,\ t = n \\ {}^{(n-1)}/_{n} & t > M,\ t > n \end{cases}, \tag{10}$$

where M denotes the memory size. We present the derivation of (10) in the Appendix A.

5 EXPERIMENTS

We perform comprehensive experiments using the feature broadcast layer on three popular citation graph datasets including Cora, Citeseer, and Pubmed (Sen et al., 2008), whose detailed statistics is listed in Appendix B. We construct two different continuum for each dataset, *i.e.*, data-incremental and class-incremental. For data-incremental tasks, all samples are randomly observed, while in class-incremental tasks, *i.e.*, each task contains one class and all samples from one task are observed before switching to the next task. For both tasks, each node can only be present to the model once, *i.e.*, a sample cannot update model parameters again once it is dropped. More implementation details are presented in Appendix C. We adopt the same evaluation metric used in (Aljundi et al., 2019b), which is the overall test accuracy after learning all tasks. We also test the feature transform layer on a social network dataset Flickr, which is presented in Appendix F due to limited pages.

As the existing graph models are difficult to directly apply to lifelong learning, we lack off-the-shelf methods to compare. Therefore, we slightly modify one of the state-of-the-art methods, GraphSAGE (Hamilton et al., 2017) to meet the lifelong learning settings. GraphSAGE needs to traverse the entire graph for every layer which is impossible for lifelong learning. To be fair, we set the neighbor sampling of GraphSAGE to available neighbors in the continuum. We also adopt the same lifelong learning method in Section 4 to alleviate its forgetting problem. Other graph models, such as GCN (Kipf & Welling, 2017), GAT (Veličković et al., 2018), and PPNP (Klicpera et al., 2019) require a full adjacency matrix, GraphSiant (Zeng et al., 2020) requires a pre-processing of the entire dataset to determine the sampling process, which cannot satisfy the requirements of lifelong learning.

Intuitively, lifelong learning has a performance upper bound given by non-lifelong learning settings. GraphSAGE achieves an accuracy of 0.850, 0.768, and 0.858 on the three datasets, respectively, while feature graph achieves a comparable accuracy of 0.857, 0.788, and 0.872, respectively. It will be shown that feature graph in lifelong learning is approaching this upper bound and has a much better performance than GraphSAGE, which verifies its effectiveness.

5.1 DATA-INCREMENTAL TASKS

The overall performance on data-incremental tasks is reported in Table 2, where all performance is an average of ten runs. For a fair comparison, we use a memory size of 100 for both methods. It can be seen that the proposed feature graph achieves an average of 5.5%, 1.9%, and 5.6% higher accuracy than GraphSAGE, respectively. It is also worth noting that both methods have a low standard deviation, which means that their performance is stable and not sensitive to the data sequence.

Table 2: Overall performance of data-incremental tasks on all datasets.

Method	Cora	Citeseer	Pubmed
GraphSAGE	0.778 ± 0.016	0.712 ± 0.015	0.816 ± 0.015
Feature Graph (ours)	$\mathbf{0.833 \pm 0.020}$	$\mathbf{0.731 \pm 0.018}$	$\mathbf{0.872 \pm 0.008}$

5.2 CLASS-INCREMENTAL TASKS

Samples in the class-incremental tasks are non-*iid* and we have no prior knowledge on the task boundary, thus it is more difficult than data-incremental tasks. We use a memory size of 200 and list their overall test accuracy in Table 3. It can be seen that the proposed feature graph achieves an average of 5.8% higher accuracy on the three datasets. This is a large margin and we expect that other customized layers as well as extensive hyper-parameter searches can further improve the performance. We also perform more tests with larger memory in Appendix D.

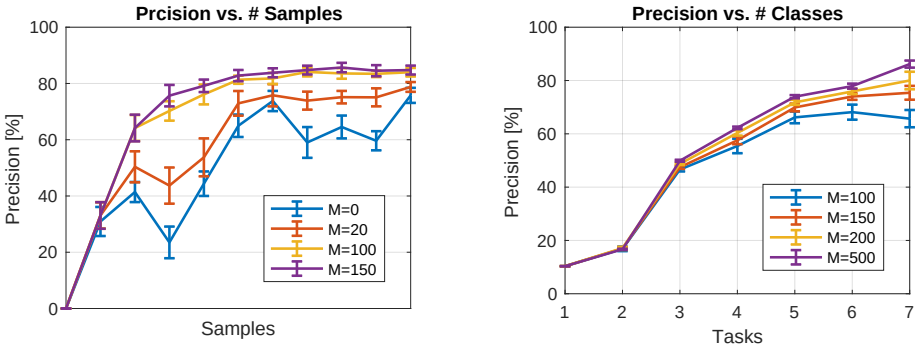
5.3 ABLATION STUDY

As reported in (Aljundi et al., 2019b), the number of exemplars from old data, which is also referred to as memory size M, may have a significant impact on lifelong learning. This section explores its

Table 3: Overall performance of class-incremental tasks on all datasets.

Method	Cora	Citeseer	Pubmed
GraphSAGE	0.686 ± 0.044	0.640 ± 0.038	0.816 ± 0.007
Feature Graph (ours)	$\mathbf{0.800} \pm \mathbf{0.033}$	$\mathbf{0.696} \pm \mathbf{0.023}$	$\mathbf{0.820} \pm \mathbf{0.018}$

influence on the feature graphs. We present the performance of both data-incremental and class-incremental tasks for the Cora dataset in Figure 2, which shows the averaged performance over ten runs. It can be seen in Figure 2a that even without any exemplars, the feature graph achieves nearly the same average precision compared to a setup with a larger memory size. This means that the feature graph can keep most of the knowledge for *iid* samples. Since the samples are non-*iid* in class-incremental learning, larger memory size will produce a better accuracy as in Figure 2b.



(a) Data-incremental learning.

(b) Class-incremental learning.

Figure 2: Model precision. (a) The feature graph is not sensitive to the memory size for data-incremental tasks. (b) Larger memory size produces higher performance on class-incremental tasks.

5.4 Efficiency

As in Table 4, our feature graph requires only $1/10$ parameters compared with GraphSAGE on the Cora dataset, which demonstrates its efficiency. As a comparison, we also list the number of parameters of GCN, which is the fastest method as reported in (Fey & Lenssen, 2019).

Table 4: The number of model parameters.

Method	GraphSAGE	GCN	Feature Graph (ours)
# Parameters	200,704	23,063	20,087

6 Conclusion

To solve the problem of lifelong graph learning, this paper introduces the feature graph as a new graph topology, which can be trained via mini-batches naturally. It takes features of a regular graph as nodes and takes the original nodes as independent graphs. This successfully converts the original node classification to graph classification and turns the problem from lifelong graph learning into regular lifelong learning. To construct the feature adjacency matrices, we accumulate the cross-correlation matrices of the connected feature vectors to model feature interactions. The comprehensive experiments show that the feature graph achieves much higher accuracy than the state-of-the-art methods in both data-incremental and class-incremental tasks. This might be one of the first practical models that bridges graph networks to lifelong learning. We note that this topic is still far from well developed, and we expect further improved performance via more customized layers and model architectures. We will also perform more experiments on other types of datasets in the future.

REFERENCES

Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 139–154, 2018.

Rahaf Aljundi, Klaas Kelchtermans, and Tinne Tuytelaars. Task-free continual learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 11254–11263, 2019a.

Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning. In *Advances in Neural Information Processing Systems*, pp. 11816–11825, 2019b.

Eden Belouadah and Adrian Popescu. Il2m: Class incremental learning with dual memory. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 583–592, 2019.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations (ICLR)*, 2014.

Francisco M Castro, Manuel J Marín-Jiménez, Nicolás Guil, Cordelia Schmid, and Karteek Alahari. End-to-end incremental learning. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 233–248, 2018.

Arslan Chaudhry, Puneet K Dokania, Thalaiyasingam Ajanthan, and Philip HS Torr. Riemannian walk for incremental learning: Understanding forgetting and intransigence. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 532–547, 2018.

Jianfei Chen, Jun Zhu, and Le Song. Stochastic training of graph convolutional networks with variance reduction. In *International Conference on Machine Learning*, 2018a.

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

Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 257–266, 2019.

Prithviraj Dhar, Rajat Vikram Singh, Kuan-Chuan Peng, Ziyan Wu, and Rama Chellappa. Learning without memorizing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5138–5146, 2019.

David L Elliott. A better activation function for artificial neural networks. Technical report, University of Maryland, 1993.

Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. In *International Conference on Learning Representations (ICLR)*, 2019.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pp. 1024–1034, 2017.

Chen He, Ruiping Wang, Shiguang Shan, and Xilin Chen. Exemplar-supported generative reproduction for class incremental learning. In *BMVC*, pp. 98, 2018.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Saihui Hou, Xinyu Pan, Chen Change Loy, Zilei Wang, and Dahua Lin. Learning a unified classifier incrementally via rebalancing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 831–839, 2019.

Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. Adaptive sampling towards fast graph representation learning. In *Advances in neural information processing systems*, pp. 4558–4567, 2018.

Ahmet Iscen, Jeffrey Zhang, Svetlana Lazebnik, and Cordelia Schmid. Memory-efficient incremental learning through feature adaptation. In *European Conference on Computer Vision (ECCV)*, 2020.

Jack Kiefer, Jacob Wolfowitz, et al. Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics*, 23(3):462–466, 1952.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference for Learning Representations (ICLR)*, 2015.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. In *International Conference on Learning Representations (ICLR)*, 2019.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems*, pp. 6467–6476, 2017.

Julian McAuley and Jure Leskovec. Image labeling on a network: using social-network metadata for image classification. In *European conference on computer vision*, pp. 828–841. Springer, 2012.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *Conference on Neural Information Processing Systems (NIPS)*, 2017.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 2001–2010, 2017.

Anthony Robins. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science*, 7(2): 123–146, 1995.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In *Advances in Neural Information Processing Systems*, pp. 2990–2999, 2017.

Konstantin Shmelkov, Cordelia Schmid, and Karteek Alahari. Incremental learning of object detectors without catastrophic forgetting. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3400–3409, 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations (ICLR)*, 2018.

Chen Wang, Jianfei Yang, Lihua Xie, and Junsong Yuan. Kervolutional neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 31–40, 2019.

Chen Wang, Wenshan Wang, Yuheng Qiu, Yafei Hu, and Sebastian Scherer. Visual memorability for robotic interestingness via unsupervised online learning. In *European Conference on Computer Vision (ECCV)*, 2020.

Max Welling. Herding dynamical weights to learn. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 1121–1128, 2009.

Chenshen Wu, Luis Herranz, Xialei Liu, Joost van de Weijer, Bogdan Raducanu, et al. Memory replay gans: Learning to generate new categories without forgetting. In *Advances In Neural Information Processing Systems*, pp. 5962–5972, 2018.

Yue Wu, Yinpeng Chen, Lijuan Wang, Yuancheng Ye, Zicheng Liu, Yandong Guo, and Yun Fu. Large scale incremental learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 374–382, 2019.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

Ye Xiang, Ying Fu, Pan Ji, and Hua Huang. Incremental learning using conditional adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 6619–6628, 2019.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*, 2018.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations (ICLR)*, 2019.

Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 974–983, 2018a.

Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Advances in neural information processing systems*, pp. 4800–4810, 2018b.

Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graph-SAINT: Graph sampling based inductive learning method. In *International Conference on Learning Representations (ICLR)*, 2020.

Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pp. 3987–3995, 2017.

Fan Zhou, Chengtai Cao, Ting Zhong, Kunpeng Zhang, Goce Trajcevski, and Ji Geng. Continual graph learning. *arXiv preprint arXiv:2003.09908*, 2020.

Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.

A SAMPLE SELECTION IN LIFELONG LEARNING

In the experiments, we adopt the simple rehearsal method with random sampling described in (Aljundi et al., 2019b). Let P be the probability that the observed items are selected at time t, thus the probability that one item is still kept in the memory after k selection is P^k. This means that earlier items have lower probability to be kept in the memory and this phenomenon was observed in (Aljundi et al., 2019b) (Sec. 4.2). To compensate for such effect and

Proposition A.1. *To ensure that all items in the continuum have the same probability to be kept in the memory at any time t, we can set the probability that the n-th item is selected at time t as*

$$P_n(t) = \begin{cases} 1 & t \leqslant M \\ {}^{M}/_{n} & t > M,\ t = n \\ {}^{(n-1)}/_{n} & t > M,\ t > n \end{cases}, \tag{11}$$

where M denotes the memory size.

Proof. The proof is obvious for $t \leqslant M$ as we only need to keep all items in the continuum. For $t > M$, the probability that the n-th item is still kept in the memory at time t is

$$\begin{aligned} P_{n,t} &= P_n(n) \cdot P_n(n+1) \cdots P_n(t-1) \cdot P_n(t) \\ &= \frac{M}{n} \cdot \frac{n}{n+1} \cdot \cdots \cdot \frac{t-2}{t-1} \cdot \frac{t-1}{t} \\ &= \frac{M}{t} \end{aligned} . \tag{12}$$

This means the probability $P_{n,t}$ is irrelevant to n and all items in the continuum share the same probability. In practice, we keep M items and sample balanced items across different classes. □

B STATISTICS OF CITATION DATASETS

We perform comprehensive experiments on popular citation graph datasets including Cora, Citeseer, and Pubmed (Sen et al., 2008). The nodes in a citation graph are articles and edges are references among the articles. The dataset statistics are listed in Table 5.

Table 5: The statistics of the datasets used for lifelong learning.

Dataset	Nodes	Edges	Classes	Features	Label Rate
Cora	2,708	5,429	7	1,433	0.421
Citeseer	3,327	4,732	6	3,703	0.337
Pubmed	19,717	44,338	3	500	0.054

C IMPLEMENTATION DETAILS

In all tasks, we adopt a feature graph model consisting of two feature broadcast layers and one fully connected layer. For hidden layers, feature channel is set as $C_{(1)} = 1$ and $C_{(2)} = 2$. We adopt the cross-entropy loss and use the softsign function $\sigma(x) = {}^{x}/_{(1+|x|)}$ as the non-linear activation (Elliott, 1993). During training, we update the model parameters for 5 times using each newly observed samples and apply the stochastic gradient descent (SGD) (Kiefer et al., 1952) for optimization, where a mini-batch size of 10 and a learning rate of 0.01 are adopted. Our implementation is based on PyTorch (Paszke et al., 2017) and tested on a Nvidia GPU of GeForce GTX 1080Ti.

D PERFORMANCE WITH LARGER MEMORY

Due to space limitation, we only show the performance using a memory size of 100 and 200 in Section 5, respectively. In this section, we further show that with a larger memory, feature networks

are able to achieve much higher performance than GraphSAGE. Table 6 and Table 7 present the average performance with 10 runs on the three citation datasets with a memory size of 500 for data-incremental and class-incremental tasks, respectively. It can be seen that the feature graph achieves an average of 3.1% and 4.8% improvements on the two tasks, respectively. This further demonstrates the effectiveness of the proposed feature graph.

Table 6: Overall performance of data-incremental tasks with memory size 500.

Method	Cora	Citeseer	Pubmed
GraphSAGE	0.854 ± 0.007	0.725 ± 0.009	0.805 ± 0.010
Feature Graph (ours)	$\mathbf{0.878} \pm \mathbf{0.010}$	$\mathbf{0.727} \pm \mathbf{0.007}$	$\mathbf{0.872} \pm \mathbf{0.007}$

Table 7: Overall performance of class-incremental tasks with memory size 500.

Method	Cora	Citeseer	Pubmed
GraphSAGE	0.802 ± 0.002	0.689 ± 0.0001	0.817 ± 0.001
Feature Graph (ours)	$\mathbf{0.862} \pm \mathbf{0.013}$	$\mathbf{0.732} \pm \mathbf{0.0140}$	$\mathbf{0.858} \pm \mathbf{0.007}$

E Feature Transform Layer

Similar to graph convolutional layer (2), the feature broadcast layer defined in Section 4.3 is able to retain the number of feature nodes in hidden layers. This is a requirement for node classification but may not be necessary for graph classification. In this section, we further define a feature transform layer which can change the number of feature nodes. Different from the feature broadcast layer, we also need to re-calculate the feature adjacency matrices from transformed neighbors. Therefore, given the feature graph $\mathcal{G}^{\mathcal{F}}$, the l-th feature transform layer can be defined as

$$\mathbf{A}_{(l)}^{\mathcal{F}}(\mathbf{x}) \triangleq \mathbf{A}_{k}^{\mathcal{F}}\left(\mathbf{x}_{(l)}, \mathbf{y}_{(l)}\right), \quad \forall \mathbf{y} \in \mathcal{N}_k(\mathbf{x}), \tag{13a}$$

$$\mathbf{x}_{(l+1)}^{\mathcal{F}} = \sigma\left(\mathbf{W}^{\mathcal{F}} \cdot \hat{\mathbf{A}}_{(l)}^{\mathcal{F}}(\mathbf{x}) \cdot \mathbf{x}_{(l)}^{\mathcal{F}}\right), \tag{13b}$$

$$\mathbf{y}_{(l+1)}^{\mathcal{F}} = \sigma\left(\mathbf{W}^{\mathcal{F}} \cdot \hat{\mathbf{A}}_{(l)}^{\mathcal{F}}(\mathbf{x}) \cdot \mathbf{y}_{(l)}^{\mathcal{F}}\right), \tag{13c}$$

where $\mathbf{W}^{\mathcal{F}} \in \mathbb{R}^{F_{(l+1)} \times F_{(l)}}$ is a learnable parameter, $\sigma(\cdot)$ is a non-linear activation function, and $\hat{\mathbf{A}}_{(l)}^{\mathcal{F}}(\mathbf{x}) \in \mathbb{R}^{F_{(l)} \times F_{(l)}}$ is the normalized feature adjacency matrix of $\mathbf{A}_{(l)}^{\mathcal{F}}(\mathbf{x})$, which is calculated from transformed feature and neighbors following (5).

F Performance of Feature Transform Layer

In this section, we further evaluate the feature graph using the feature transform layer. We construct the network by concatenating two feature transform layers with one fully connected layer. We test the network on a larger dataset, Flickr (McAuley & Leskovec, 2012), which is listed in Table 8. This dataset is built by forming links between images sharing common metadata from Flickr. Edges are formed between images from the same location, submitted to the same gallery, group, or set, etc. We update the model once using each newly available samples and apply the Adam method for training (Kingma & Ba, 2015), where a mini-batch size of 10 and a learning rate of 0.001 are adopted.

Table 8: The statistics of the Flickr dataset.

Dataset	Nodes	Edges	Classes	Features	Label Rate
Flickr	89,250	899,756	7	500	0.5

It is worth noting that the non-lifelong learning method GraphSIANT (Zeng et al., 2020) achieves an upper bound accuracy of 0.51, which indicates that Flickr is very difficult. Nevertheless, it can

be seen in Table 9 that our feature graph achieves an average accuracy of 0.470 and 0.468 even with a zero memory size for the data-incremental and class-incremental tasks, respectively. This further demonstrates the effectiveness of the proposed feature graph topology. We observe an overfitting for larger memory on the Flickr dataset. This is because we adopt a simple updating strategy using the memory samples and larger memory results in more update steps. We believe that a well-designed updating strategy is able to further improve the performance, but it is out the scope of this paper.

Table 9: Overall performance of the feature graph on the Flickr dataset.

Memory	M=0	M=10	M=20
Data-incremental	0.470 ± 0.007	0.464 ± 0.008	0.466 ± 0.004
Class-incremental	0.468 ± 0.008	0.474 ± 0.003	0.447 ± 0.009