Lifelong Graph Learning

Chen Wang
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
chenwang@dr.com

Yuheng Qiu
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
yuhengq@andrew.cmu.edu

Dasong Gao
Machine Learning Department
Carnegie Mellon University
Pittsburgh, PA 15213
dasongg@andrew.cmu.edu

Sebastian Scherer
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
basti@andrew.cmu.edu

Abstract

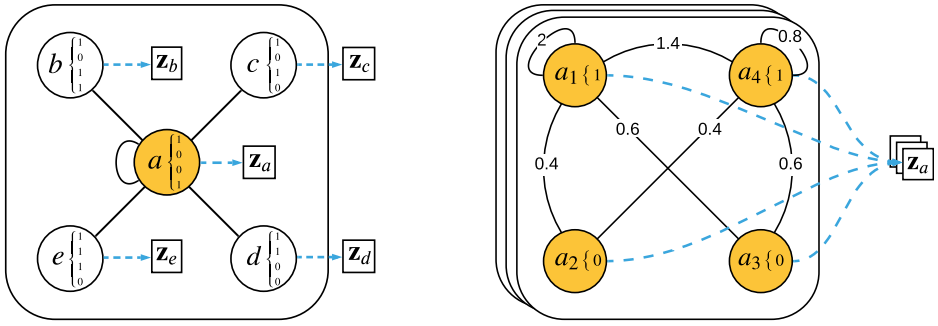
Graph neural networks (GNNs) are powerful models for many graph-structured tasks. Existing models often assume that a complete structure of a graph is available during training, however, in practice, graph-structured data is usually formed in a streaming fashion, so that learning a graph continuously is often necessary. In this paper, we aim to bridge GNN to lifelong learning by converting a graph problem to a regular learning problem, so that GNN is able to inherit the lifelong learning techniques developed for convolutional neural networks (CNNs). To this end, we propose a new graph topology based on feature cross-correlation, 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 independent training samples. In the experiments, we demonstrate the efficiency and effectiveness of feature graph networks (FGN) by continuously learning a sequence of classical graph datasets. We also show that FGN achieves superior performance in human action recognition with distributed streaming signals for wearable devices.

1 Introduction

Graph neural networks (GNN) have received increasing attention and proved to be useful for many tasks with graph-structured data, such as citation, social, and protein networks [1]. However, graph data is usually formed in a streaming fashion and real-world data are continuously evolving over time, thus learning a streaming graph is expected in many cases [2]. For example, in a social network, the number of users often grows over time and we expect that the model can learn continuously with new users. In this paper, we extend graph neural networks to lifelong learning, which is also known as continual or incremental learning [3]. However, lifelong learning easily suffers from "catastrophic forgetting" if the models are simply updated for new samples via back-propagation [4].

Although some strategies have been developed to alleviate the forgetting problem for convolutional neural networks (CNN), lifelong learning is still difficult for graph networks. This is because existing graph models such as graph convolutional networks (GCN) [5] require the entire graph for training. This is infeasible for lifelong learning, since the graph size can increase over time and we have to drop off old data or samples 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 [6–8], since they require a pre-processing of the entire graph [8].

Preprint. Under review.



(a) Regular graph $\mathcal{G}$.

(b) Feature graph $\mathcal{G}^{\mathcal{F}}$.

Figure 1: A feature graph takes the features as nodes and turns nodes into graphs, resulting in a graph predictor instead of the original node predictor. This makes the lifelong learning techniques for CNN applicable to GNN, as the increasing nodes in a regular graph become training samples. Take the node a with label $\mathbf{z}_a$ in the regular graph $\mathcal{G}$ as an example, its features $\mathbf{x}_a = [1, 0, 0, 1]$ are nodes $\{a_1, a_2, a_3, a_4\}$ in feature graph $\mathcal{G}_a^{\mathcal{F}}$. The feature adjacency is established via feature cross-correlation between a and its neighbors $\mathcal{N}(a) = \{a, b, c, d, e\}$ to model feature "interaction".

Recall that regular CNNs are trained in a mini-batch manner where the model can take samples as independent inputs [9]. Our question is: can we convert a graph task 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 is not straightforward as node connections cannot be modeled by a regular CNN-like classification model. To solve this problem, we propose to construct a new graph topology, the feature graph in Fig. 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 and the increasing nodes become independent training samples, thus the feature graphs can be trained in a regular mini-batch manner.

The contribution of this paper includes: (1) We introduce a new graph topology, *i.e.*, feature graph, to convert a problem of growing graph to increasing training samples, which makes lifelong learning techniques applicable to graph neural networks; (2) We take the cross-correlation of neighbor features as the feature adjacency matrix, which is able to explicitly model feature "interaction", that is crucial for many graph-structured tasks; (3) Feature graph is of constant computational complexity with the increased learning tasks. We demonstrate its efficiency and effectiveness by applying it to classical graph datasets; (4) We further demonstrate its superiority by applying it to a practical application, *i.e.*, distributed human action recognition based on subgraph classification using wearable devices.

2 Related Work

2.1 Lifelong Learning

Non-rehearsal Methods Lifelong learning methods in this category do not preserve any old data. To alleviate the forgetting problem, progressive neural networks [10] leveraged prior knowledge via lateral connections to previously learned features. Learning without forgetting (LwF) [11] introduced a knowledge distillation loss [12] 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 [13]. Learning without memorizing (LwM) [14] 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 [15] remembered old tasks by slowing down learning on important weights. RWalk [16] generalized EWC and improved weight consolidation by adding a KL-divergence-based regularization. Memory aware synapses (MAS) [17] computed an importance value for each parameter in an unsupervised manner based on the sensitivity of output function to parameter changes. [18] presented an embedding framework for dynamic attributed network based on parameter regularization. A sparse writing protocol is introduced to a memory module [19], ensuring that only a few memory spaces is affected during training.

Rehearsal Methods Rehearsal lifelong learning methods can be roughly divided into rehearsal with synthetic data or rehearsal with exemplars from old data [20]. To ensure that the loss of exemplars does not increase, gradient episodic memory (GEM) [3] introduced orientation constraints during gradient updates. Inspired by GEM, [21] selected exemplars with a maximal cosine similarity of the gradient orientation. iCaRL [22] preserved a subset of images with a herding algorithm [23] and included the subset when updating the network for new classes. EEIL [24] extended iCaRL by learning the classifier in an end-to-end manner. [25] further extended iCaRL and discovered that updating the model with class-balanced exemplars further improved the performance. Similarly, [26, 27] further added constraints to the loss function to mitigate the effect of imbalance. To reduce the memory consumption of exemplars, [28] applied the distillation loss to feature space without having to access to the corresponding images. Rehearsal approaches with synthetic data based on generative adversary networks (GAN) were used to reduce the dependence on old data [29–32].

2.2 Graph Neural Networks

Graph neural networks have been widely used to solve problems about graph-structured data [33]. The spectral network was the first to extend convolution to the graph domain [34]. Later the graph convolutional network (GCN) [5] 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. To identify the importance of neighborhood features, graph attention network (GAT) [35] added an attention mechanism into GCN, which further improved the performance on citation networks and the protein-protein interaction dataset.

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, a sampling method, SAGE [6] is introduced to learn a function to generate node embedding by sampling and aggregating neighborhood features. JK-Net [36] followed the same sampling strategy and demonstrated a significant accuracy improvement on GCN with jumping connections. DiffPool [37] learned a differentiable soft cluster assignment to map nodes to a set of clusters, which then formed a coarsened input for the next layer. [38] designed a training strategy that relied on harder-and-harder training examples to improve the robustness and convergence speed of the model. S-GCN [39] developed control variate-based algorithms to sample small neighbor sizes and used historical activation in the previous layer to avoid re-evaluation.

FastGCN [7] applied importance sampling to reduce variance and performed node sampling for each layer independently, which resulted in a constant sample size in all layers. [40] 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 [41] sampled a block of nodes in a dense subgraph identified by a clustering algorithm and restricted the neighborhood search within the subgraph. GraphSiant [8] constructed mini-batches by sampling the training graph.

Nevertheless, most of the sampling techniques still require a pre-processing of the entire graph to determine the 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 its original meaning. We noticed that there is some recent work focusing on continuously learning a graph problem, but they have different formulation. For example, several exemplar selection methods are tested in ER-GNN [42]. A weight preserving method is introduced to growing graph [43]. A combined strategy of regularization and data rehearsal is introduced to streaming graphs in [2]. To overcome the incomplete structure, learning a temporal graph in sliding window is introduced in [44].

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 the set of nodes and $\mathcal{E} \subseteq \left\{\{v_a, v_b\} \,|(v_a, v_b) \in \mathcal{V}^2\right\}$ is the set of edges. Each node $v \in \mathcal{V}$ is associated with a target $\mathbf{z}_v \in \mathcal{Z}$ 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 vector $\mathbf{w}_e \in \mathcal{W} \subset \mathbb{R}^W$. In regular graph learning, we aim to learn a predictor f to associate a node $\mathbf{x}_v, v \in \mathcal{V}'$ with a target $\mathbf{z}_v$, given graph $\mathcal{G}$, node features $\mathcal{X}$, edge 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 an edge 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 edges. 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 of a single learning task. In lifelong graph learning, we will observe, item by item, the continuum of the graph-structured data as

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

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. The task descriptors t_i is defined for compatibility with lifelong learning that requires it [3], but it is not used in this paper.

Note that samples are **not** drawn locally identically and independently distributed (*iid*) from a fixed probability distribution, since we don't know the task boundary. In the continuum, we only know the label of $\mathbf{x}_i$, but have no information about the labels of its neighbors $\mathcal{N}(\mathbf{x}_i)$. Note that items in (1) are unavailable once they are observed and dropped off. This is in contrast to the settings in [2], where all historical data are available during training. As shown in the experiments, lifelong graph learning in practice often requires that the number of GNN layers L is larger than the availability of K-hop neighbors, *i.e.*, $L > K$, which also leads many existing graph models inapplicable.

4 Feature Graph Network

To better show the relationship with a regular graph, we first review GCN [5]. Given an 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 $\mathbf{x}_i \in \mathbf{R}^F, \mathbf{X} \in \mathbb{R}^{N \times F}$. The l-th graph convolutional layer is 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 a self-connected symmetrically normalized adjacency matrix of $\mathbf{A}$ [5]. For unweighted graph, $\mathbf{A}_{[i,j]} = 1$ if $(i, j) \in \mathcal{E}$ and $\mathbf{A}_{[i,j]} = 0$ otherwise. A graph convolutional layer doesn't change the number of nodes (rows of $\mathbf{X}$), but change the node feature dimension from $F_{(l)}$ 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 be presented to the model. This limits its usage to large graphs and makes it inapplicable to lifelong learning either. As aforementioned, to scale GCN to large graphs, many sampling techniques are proposed but requires a pre-processing of the entire datasets [8], thus most of them are still difficult to directly apply to lifelong learning. We argue that such difficulty is caused by the inappropriate graph topology and introduce the feature graphs starting from the definition of feature nodes.

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 define the feature nodes for feature graph as

$$\mathcal{V}^{\mathcal{F}} = \left\{\mathbf{x}_1^{\mathcal{F}}, \mathbf{x}_2^{\mathcal{F}}, \cdots, \mathbf{x}_i^{\mathcal{F}}, \cdots, \mathbf{x}_F^{\mathcal{F}}\right\}. \tag{3}$$

In this way, for each node $v \in \mathcal{V}$, we have an associated feature graph $\mathcal{G}^{\mathcal{F}}$. We next establish their relationship by defining the feature adjacency matrices via feature cross-correlation.

Table 1: The relationship of conversion from a regular graph to 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	Growing adjacency matrix	$\mapsto$	Multiple adjacency matrices
Graph	$\mapsto$	Samples	Dynamic graph	$\mapsto$	Multiple samples

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:

$$\mathbf{A}_{k,c}^{\mathcal{F}}(\mathbf{x}) \triangleq \operatorname{sgnroot}\left(\mathbb{E}\left[w_{x,y}\mathbf{x}_{[:,c]}\mathbf{y}_{[:,c]}^{\mathrm{T}}\right]\right), \quad \text{subject to} \quad \forall \mathbf{y} \in \mathcal{N}_k(\mathbf{x}), \quad c = 1, 2, \cdots, C, \tag{4}$$

where $w_{x,y}$ is the associated edge weight and $\operatorname{sgnroot}(x) = \operatorname{sign}(x)\sqrt{(|x|)}$ is introduced to retain the magnitude of node features and the sign of their inner products. Note that $\mathbf{x}_{[:,c]}$ in (4) 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 (4) is approximated by averaging over the observed neighbors in the continuum.

$$\mathbb{E}\left[w_{x,y}\mathbf{x}_{[:,c]}\mathbf{y}_{[:,c]}^{\mathrm{T}}\right] \approx \frac{\sum_{\mathbf{y}\in\mathcal{N}_k(\mathbf{x})} w_{x,y}\mathbf{x}_{[:,c]}\mathbf{y}_{[:,c]}^{\mathrm{T}}}{|\mathcal{N}_k(\mathbf{x})|}, \tag{5}$$

where we need to change $\mathbf{x}_{[:,c]}\mathbf{y}_{[:,c]}^{\mathrm{T}}$ to $(\mathbf{x}_{[:,c]}\mathbf{y}_{[:,c]}^{\mathrm{T}} + \mathbf{y}_{[:,c]}\mathbf{x}_{[:,c]}^{\mathrm{T}})$ for undirected graphs. Therefore, the feature adjacency matrix can be constructed online via (5), together with the availability of new samples in the continuum. In this way, the continuum (1) is converted to

$$(\mathcal{G}_1^{\mathcal{F}}, t_1, \mathbf{z}_1), \ldots, (\mathcal{G}_i^{\mathcal{F}}, t_i, \mathbf{z}_i), \ldots, (\mathcal{G}_N^{\mathcal{F}}, t_N, \mathbf{z}_N) \tag{6}$$

where $\mathcal{G}^{\mathcal{F}} = \left(\mathcal{V}^{\mathcal{F}}, \mathbf{A}^{\mathcal{F}}(\mathbf{x})\right)$. This means that our objective of learning a node predictor becomes learning a graph predictor $f^{\mathcal{F}}: \mathcal{G}^{\mathcal{F}} \times \mathcal{T} \mapsto \mathcal{Z}$ 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 feature graphs in the new continuum (6) are still non-*iid*, and each feature graph is associated with a unique feature adjacency matrix.

In this way, a growing adjacency matrix is converted to multiple small adjacency matrices of feature graphs. Hence, a lifelong graph learning problem becomes a regular lifelong learning problem similar to [3] and the problem of increasing nodes in graph learning can be solved by applying lifelong learning to the continuum. To be more clear, we list their detailed relationship in Table 1, where the arrow $\mapsto$ refers to the conversion from a regular graph to feature graphs.

4.3 Feature Graph Layers

Since feature graph is a new topology given by feature adjacency matrix, we are able to define many different types of layers. In this section we present three types of layers.

4.3.1 Feature Broadcast Layer

Inspired by GCN, the l-th broadcast layer is defined as

$$\mathbf{x}_{(l+1)}^{\mathcal{F}} = \sigma\left(\hat{\mathbf{A}}_k^{\mathcal{F}} \cdot \mathbf{x}_{(l)}^{\mathcal{F}} \cdot \mathbf{W}^{\mathcal{F}}\right), \tag{7}$$

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. For simplicity, the channel c is left out in (7) and the layer channel is broadcasted independently. It is worth noting that, although the definition of (7) appears similar to (2), they have different dimension and represent different meanings.

4.3.2 Feature Transform Layer

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. In this section, we define a feature transform layer which can change the number of feature nodes and will be helpful for further reducing the number of learnable parameters. Different from the feature broadcast layer, we 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

$$\mathbf{A}^{\mathcal{F}}_{(l)}(\mathbf{x}) \triangleq \mathbf{A}^{\mathcal{F}}_{k}\left(\mathbf{x}_{(l)}, \mathbf{y}_{(l)}\right), \quad \forall \mathbf{y} \in \mathcal{N}_k(\mathbf{x}), \tag{8a}$$

$$\mathbf{x}^{\mathcal{F}}_{(l+1)} = \sigma\left(\mathbf{W}^{\mathcal{F}} \cdot \hat{\mathbf{A}}^{\mathcal{F}}_{(l)}(\mathbf{x}) \cdot \mathbf{x}^{\mathcal{F}}_{(l)}\right), \quad \mathbf{y}^{\mathcal{F}}_{(l+1)} = \sigma\left(\mathbf{W}^{\mathcal{F}} \cdot \hat{\mathbf{A}}^{\mathcal{F}}_{(l)}(\mathbf{x}) \cdot \mathbf{y}^{\mathcal{F}}_{(l)}\right), \tag{8b}$$

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 $\hat{\mathbf{A}}^{\mathcal{F}}_{(l)}(\mathbf{x}) \in \mathbb{R}^{F_{(l)} \times F_{(l)}}$ is the self-connected normalized feature adjacency matrix of $\mathbf{A}^{\mathcal{F}}_{(l)}(\mathbf{x})$, which is calculated from transformed feature and neighbors following (4). In some cases, the node features can be smoothed due to graph propagation, hence we can replace $\hat{\mathbf{A}}^{\mathcal{F}}_{(l)}(\mathbf{x}) \cdot \mathbf{y}^{\mathcal{F}}_{(l)}$ in (8b) to $[\hat{\mathbf{A}}^{\mathcal{F}}_{(l)}(\mathbf{x}) \cdot \mathbf{y}^{\mathcal{F}}_{(l)}, \mathbf{y}^{\mathcal{F}}_{(l)}]$ to concatenate input features to prevent over-smoothing.

4.3.3 Feature Attention Layer

In the cases that a graph is fully connected or the edges have no weights, $w_{x,y}$ in (4) will be not well defined. Prior methods often rely on an attention mechanism, *e.g.*, GAT [35], to focus on the most important neighbors. Inspired by this, we can define the edge weights as an attention in (9).

$$w_{x,y} = \frac{\exp(e_{x,y})}{\sum_{z \in \mathcal{N}(z)} \exp(e_{x,z})}, \quad e_{x,y} = \mathrm{LeakyReLU}(\mathbf{a}_x^T \mathbf{x} + \mathbf{a}_y^T \mathbf{y} + b), \tag{9}$$

where $\mathbf{a}_x, \mathbf{a}_y \in \mathbb{R}^F, b \in \mathbb{R}$ are learnable attention parameters. Note that we can construct other types of layers based on the topology $\mathcal{G}^{\mathcal{F}}$, *e.g.*, extending convolution to kervolution [45] to improve the model expressivity or combining the pagerank algorithm [46] to further reduce feature over-smoothness. In this paper, we will mainly demonstrate the effectiveness of the above three layers.

4.4 Analysis and Computational Complexity

We next provide an intuitive explanation for feature graphs. A feature graph doesn't retain the physical meaning of its regular graph. Take a social network as an example, in a regular graph, each user is a node and user connections are edges. In feature graphs, each user is a graph, while the user features are nodes, including the users' age, gender, *etc.* Therefore, user behavior prediction becomes a problem of graph classification from node classification and the increasing users simply become multiple training samples. Since the number of user features is more stable than the number of users, the size of a feature graph is more stable than its regular graph. This simplifies the learning process dramatically since the lifelong graph learning can be solved by regular lifelong learning techniques.

On the other hand, a regular graph usually assumes that some useful information of a node is often encoded into its neighbors, thus graph propagation is able to improve the model performance. A feature graph has the same assumption. However, feature graph doesn't directly propagate the neighbor features, but encode the neighbors into the feature adjacency matrices (4). Regardless of the nonlinear function, existing methods [5, 8, 35, 46] propagate neighbor features as $\hat{\mathbf{A}}\mathbf{X}$, where $\mathbf{X} = [\mathbf{x}_1, \cdots, \mathbf{x}_n]$, which means that they can only propagate neighbors in an element-wise manner, as nodes in the next layer are weighted average of the current nodes. In contrast, each feature graph propagate features via $\hat{\mathbf{A}}^{\mathcal{F}}\mathbf{x}^{\mathcal{F}}$, which explicitly model "interaction" between features. In this sense, feature graph doesn't lose information of edges but encode them into the feature adjacency. This explains the phenomenon that it also outperforms the graph models in some cases of regular graph learning presented in Section 5. Take a citation graph as an example, a keyword in an article may have an impact on another keyword in the article citing the former one. However, element-wise graph propagation like (2) [5, 8, 35, 46], cannot explicitly model this relationship. Therefore, feature graph will also work for regular graph learning, although we mainly focus on lifelong graph learning.

Feature graphs have a low computational complexity. Concretely, the complexity for calculating the feature adjacency is $\mathcal{O}(nF^2_{(l)})$, where n is the expected number of neighbors of each node in the continuum. Therefore, regardless of the number of layers which is a constant for specific models,

Table 2: Overall performance comparison on all the datasets.[†]

Method	Task[‡]	Cora	Citeseer	Pubmed	ogbn-arXiv
MLP	R	0.673	0.702	0.832	0.462
GCN		0.850	0.768	0.868	0.557
SAGE		0.850	0.768	0.858	0.615
GAT		<u>0.877</u>	0.776	<u>0.872</u>	<u>0.619</u>
APPNP		0.883	<u>0.777</u>	0.870	0.615
FGN		**0.887**	**0.785**	**0.884**	**0.631**
MLP	D	0.652 ± 0.006	0.707 ± 0.008	0.812 ± 0.006	0.349 ± 0.006
GCN		0.827 ± 0.003	<u>0.760 ± 0.004</u>	0.845 ± 0.002	0.397 ± 0.012
SAGE		0.778 ± 0.016	0.712 ± 0.015	0.816 ± 0.015	<u>0.531 ± 0.006</u>
GAT		0.857 ± 0.007	0.735 ± 0.009	0.847 ± 0.002	0.512 ± 0.003
APPNP		<u>0.861 ± 0.001</u>	0.725 ± 0.016	<u>0.851 ± 0.003</u>	0.511 ± 0.002
FGN		$\mathbf{0.870 \pm 0.011}$	$\mathbf{0.762 \pm 0.003}$	$\mathbf{0.872 \pm 0.009}$	$\mathbf{0.555 \pm 0.010}$
MLP	C	0.558 ± 0.010	0.597 ± 0.012	0.816 ± 0.013	0.237 ± 0.011
GCN		0.796 ± 0.022	0.695 ± 0.003	<u>0.851 ± 0.012</u>	0.328 ± 0.011
SAGE		0.686 ± 0.044	0.640 ± 0.038	0.816 ± 0.007	<u>0.455 ± 0.003</u>
GAT		<u>0.833 ± 0.032</u>	<u>0.698 ± 0.009</u>	0.833 ± 0.006	0.409 ± 0.006
APPNP		0.818 ± 0.017	0.689 ± 0.003	0.844 ± 0.013	0.423 ± 0.007
FGN		$\mathbf{0.853 \pm 0.014}$	$\mathbf{0.733 \pm 0.009}$	$\mathbf{0.857 \pm 0.003}$	$\mathbf{0.483 \pm 0.006}$

[†]We denote the best performance in **bold** and second best with <u>underline</u> for each task.
[‡]Task of "R", "D", "C" denote regular, data-incremental, and class-incremental learning, respectively.

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.

5 Experiments

Implementation Details We take the one-hot vector as the target vector $\mathbf{z}_i$, adopt the cross-entropy loss, and use the softsign [47] function $\sigma(x) = x/(1+|x|)$ as the non-linear activation. Our implementation is based on PyTorch [48]. We perform comprehensive tests on popular graph datasets including citation graph Cora, Citeseer, Pubmed [49], and ogbn-arXiv [50]. We list more details for these datasets in Appendix A. We use two feature broadcast layers (7) for Cora, Citeseer, and Pubmed. Feature channel is set as $C_{(1)} = 1$ and $C_{(2)} = 2$ for hidden layers. For obgn-arXiv, we found that node features can be easily smoothed due to multiple feature propagation, hence we use the feature transform layers to concatenate its input features. We adopt the SGD [51] optimizer for Cora, Citeseer, and Pubmed and the Adam optimizer [52] and obgn-arXiv. We choose the most popular graph models including GCN [5], GAT [35], SAGE [6], and APPNP [46] as our baseline. The other methods, such as SIANT [44] require pre-processing entire dataset, thus it cannot be applied to lifelong learning. The overall test accuracy after learning all tasks is adopted as the evaluation metric [21]. For each dataset, we construct two different continuum, *i.e.*, data-incremental and class-incremental. For data-incremental tasks, all samples are streamed randomly, while in class-incremental tasks, *i.e.*, all samples from one class are streamed before switching to the next class. In both tasks, each node can only be present to the model once, *i.e.*, a sample cannot update model parameters again once dropped.

Sequence Invariant Sampling We adopt the lifelong learning method described in [21] with a minor improvement for all the baseline models. As reported in the Section 4.2 of [21], it tends to select less earlier items in the continuum and this has a negative effect on remembering earlier knowledge. We find that this is because that a uniform sampling is adopted for all items. To compensate for such effect, we set a customized selection probability for different items and present it in the Appendix B.

Performance Lifelong learning has a performance upper bound given by the regular learning settings, thus the performance in regular learning is also an important indicator for the effectiveness of feature graph. We present the overall performance of regular learning in Table 2 and denote this task as "R", where the same dataset settings are adopted for different models. To demonstrate the

necessity of graph models, we also give the performance of MLP, which neglects of the graph edges. It can be seen that feature graph achieves a comparable overall accuracy in all the datasets. This means that feature graph may also be useful for regular graph learning, although we focus on lifelong graph learning in this paper. We next show that feature graph in lifelong learning is approaching this upper bound and dramatically alleviate the issue of "catastrophic forgetting".

The overall averaged performance on the data-incremental tasks is reported as task of "D" in Table 2, where all performance is an average of three runs. We use a memory size of 500 for the datasets Cora, Citeseer and Pubmed, and a memory size of 512 for ogbn-arXiv. It is also worth noting that we have a low standard deviation, which means that its performance is stable and not sensitive to the model initialization. Samples in this class-incremental tasks are non-*iid* and we have no prior knowledge on the task boundary, thus it is generally *more difficult* than data-incremental tasks. The overall test accuracy on class-incremental tasks are listed as "C" in Table 2. Similar to the data-incremental tasks, we use a memory size of 500 for Cora, Pubmed, and Citeseer, and a memory size of 4096 for ogbn-arXix, which is grouped into 5 tasks, each of which contains 8 classes. Despite its difficult, FGN is still able to obtain meaningful performance on ogbn-arXix in task "C", compared to other state-of-the-art methods, which verifies its superiority in lifelong graph learning. We further report the forgetting rate in the task "C" for the best models in Table 3, which is the averaged class precision dropping compared to the regular learning. It can be seen that FGN achieves the least forgetting rate, which further demonstrates its effectiveness in lifelong graph learning.

Table 3: Average class forgetting rate in task "C" (%).

Model	Cora	Citeseer	Pubmed	ogbn-arXiv
APPNP	6.38	8.03	3.61	16.3
GAT	5.50	7.05	3.50	17.7
FGN	**1.35**	**0.41**	**3.49**	**14.8**

6 Distributed Human Action Recognition with Wearable Devices

Implementation Details To demonstrated the flexibility of FGN, we apply it to an application, *i.e.*, distributed human action recognition using wearable motion sensor networks in Fig. 2a (adapted from [53]). Five sensors, each of which consists of a triaxial accelerometer and a biaxial gyroscope, are located at the left and right forearms, waist, left and right ankles, respectively. Each sensor produces 5 data streams and totally 5×5 data streams is available. The stream is recorded at 30Hz and is comprised of human subjects with ages from 19 to 75 and 13 daily action categories, including rest at standing (ReSt), rest at sitting (ReSi), rest at lying (ReLi), walk forward (WaFo), walk forward left-circle (WaLe), walk forward right-circle (WaRi), turn left (TuLe), turn right (TuRi), go upstairs (Up), go downstairs (Down), jog (Jog), jump (Jump), and push wheelchair (Push).

We take each 25 sequential data points from a single sensor as a node and perform recognition for every 50 sequential points (1.67s) shown in Fig. 2b, which is also adopted by [54]. This results in a temporal growing (streaming) graph, where each node is associated with a multi-channel ($C = 5$) 1-D signal $\mathbf{x}_t^i \in \mathbb{R}^{25\times 5}$, where $i = 1, 2 \ldots 5$ denotes the i-th sensor and $t = 1, 2 \ldots T$ is the time index. In this way, the problem of distributed human action recognition becomes a problem of sub-graph (10 nodes) classification and lifelong human action recognition becomes a lifelong graph learning problem. In practice, the temporal growing graph can only be learned sequentially, thus we take the first 80% of each sequence for training and the remaining 20% for testing. Specifically, we define the radius of neighborhood as the temporal distance. Therefore, all the nodes at the same instant are 1-hop neighbors of each other. For each feature graph we have $K = 2$ in the continuum (1). We construct FGN using two feature transform layers (8) with attention weights (9) to construct the sub-graph embedding. For fairness, we use $C_{(1)} = 5$, $F_{(1)} = 25$, $C_{(2)} = 32$, and $F_{(2)} = 12$ for all models. During training, we find that GCN, APPNP are able to obtain best overall performance using the SGD optimizer, while MLP, GAT, and FGN performed the best using the Adam optimizer.

Performance We list the overall performance of regular and lifelong learning in Table 4, which is obtained from an average of three runs. It can be seen that FGN achieves the best overall performance in regular learning and a much higher performance in lifelong learning compared to the state-of-the-art methods. This means that FGN has a lower forgetting rate, demonstrating its effectiveness.

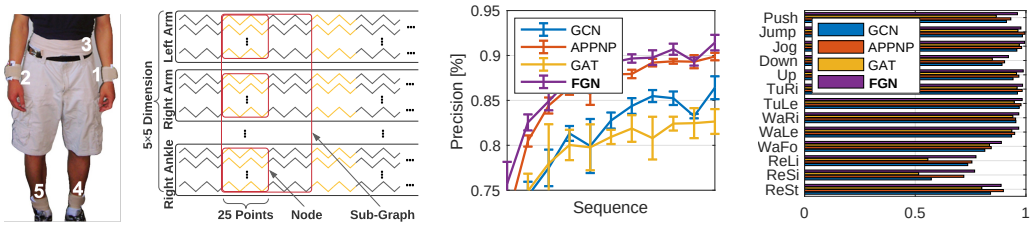
We present the overall test accuracy during the lifelong learning process in Fig. 2c. It can be seen that FGN has a much higher and stable performance than all the other methods. We also shown their final per-class precision in Fig. 2d, which indicates that FGN achieves a much higher performance in

Table 4: Overall performance on the action recognition of streaming graph with wearable sensors.

Task	MLP	GCN	APPNP	GAT	**FGN**
Regular	0.645 ± 0.046	0.881 ± 0.016	0.942 ± 0.006	0.911 ± 0.004	$\mathbf{0.954} \pm \mathbf{0.004}$
Lifelong	-[†]	0.864 ± 0.013	0.898 ± 0.004	0.826 ± 0.014	$\mathbf{0.914} \pm \mathbf{0.009}$

[†]The notation "-" means that we cannot obtain meaningful performance on this task.

nearly all the categories. Note that all models have a relatively low performance in the three activities including ReSt, ReSi, and ReLi. This is because their signals are relatively flat and similar due to their static posture. This phenomenon is also observed by a template matching method KCC [54]. Note that we don't directly compare against to KCC, since KCC mainly performs matching for single subject, while we aim for prediction across different subjects.



(a) Sensors. (b) The temporal growing graph. (c) Lifelong learning. (d) Per-class precision.

Figure 2: The wearable sensor networks, the streaming graph, and the performance comparison.

We argue that the superior performance of FGN is because of that FGN is able to explicitly model the relationship of different features. Take the walking action as an example, it is well known that the movement of a left arm is always related to the movement of the right leg. Such information can be explicitly modeled by the cross-correlation by the feature adjacency matrix. Moreover, it is also able to model the feature relationship at different time step, *e.g.*, the movement of a left arm at time t is also related to the movement of the right arm at time $t+1$. As aforementioned, FGN can explicitly learn such relationship with $k = 2$ in (4), while the other methods can't directly do this. Besides, FGN also achieves the least forgetting rate in Table 5, which is calculated from the difference between the final and best performance during the entire learning process.

Table 5: Backward maximum forgetting rate of all models on all actions (%).

Model	ReSt	ReSi	ReLi	WaFo	WaLe	WaRi	TuLe	TuRi	Up	Down	Jog	Jump	Push	**Overall**
GCN	5.34	12.49	0.99	1.32	2.88	0.59	0.89	1.58	1.30	2.81	1.24	0.26	5.31	2.85
APPNP	0.81	2.57	7.17	1.37	2.70	0.80	0.94	0.44	0.11	3.97	0.99	0.00	3.66	1.96
GAT	1.34	6.12	4.23	0.00	2.79	0.76	0.82	0.83	0.81	2.27	0.62	0.87	4.79	2.02
FGN	0.19	0.56	3.32	0.95	2.67	0.53	1.15	1.04	0.00	3.52	0.46	1.02	2.28	**1.36**

7 Conclusion

In this paper, we focus on the problem of lifelong graph learning and propose the feature graph as a new graph topology, which solves the challenge of increasing nodes in a streaming graph. It takes features of a regular graph as nodes and takes the nodes as independent graphs. To construct the feature adjacency matrices, we accumulate the cross-correlation matrices of the connected feature vectors to model feature interactions. This successfully converts the original node classification to graph classification and turns the problem from lifelong graph learning into regular lifelong learning. The comprehensive experiments show that the feature graph achieves superior performance in both data-incremental and class-incremental tasks. The practical applications on distributed action recognition also demonstrates its superiority in lifelong graph learning.

Acknowledgment

This work was sponsored by ONR grant #N0014-19-1-2266.

References

[1] 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.

[2] Junshan Wang, Guojie Song, Yi Wu, and Liang Wang. Streaming graph neural networks via continual learning. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 1515–1524, 2020.

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

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

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

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

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

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

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

[10] 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.

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

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

[13] 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*, pages 3400–3409, 2017.

[14] 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*, pages 5138–5146, 2019.

[15] 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.

[16] 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)*, pages 532–547, 2018.

[17] 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)*, pages 139–154, 2018.

[18] Hao Wei, Guyu Hu, Wei Bai, Shiming Xia, and Zhisong Pan. Lifelong representation learning in dynamic attributed networks. *Neurocomputing*, 358:1–9, 2019.

[19] 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.

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

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

[22] 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*, pages 2001–2010, 2017.

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

[24] 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)*, pages 233–248, 2018.

[25] 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*, pages 374–382, 2019.

[26] 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)*, pages 831–839, 2019.

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

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

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

[30] 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*, pages 5962–5972, 2018.

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

[32] 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*, pages 6619–6628, 2019.

[33] 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.

[34] 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.

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

[36] 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.

[37] 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*, pages 4800–4810, 2018.

[38] 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*, pages 974–983, 2018.

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

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

[41] 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*, pages 257–266, 2019.

[42] Fan Zhou and Chengtai Cao. Overcoming catastrophic forgetting in graph neural networks with experience replay. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2021.

[43] Huihui Liu, Yiding Yang, and Xinchao Wang. Overcoming catastrophic forgetting in graph neural networks. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2021.

[44] Lukas Galke, Iacopo Vagliano, and Ansgar Scherp. Incremental training of graph neural networks on temporal graphs under distribution shift. *arXiv preprint arXiv:2006.14422*, 2020.

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

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

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

[48] 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.

[49] 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.

[50] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[51] 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.

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

[53] Allen Y Yang, Roozbeh Jafari, S Shankar Sastry, and Ruzena Bajcsy. Distributed recognition of human actions using wearable motion sensor networks. *Journal of Ambient Intelligence and Smart Environments*, 1(2):103–115, 2009.

[54] Chen Wang, Le Zhang, Lihua Xie, and Junsong Yuan. Kernel cross-correlator. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

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

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

A Dataset Details

We perform comprehensive experiments on popular graph datasets including citation graph Cora, Citeseer, Pubmed, and ogbn-arXiv [49, 50, 55]. Their statistics are listed in Table 6.

Table 6: The statistics of the datasets used for lifelong learning.

Dataset	Nodes	Edges	Classes	Features	Labels
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
obgn-arXiv	169,343	1,166,243	40	128	0.5

B Proof of Sequence Invariant Sampling

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 explains that earlier items have lower probability to be kept in the memory and this phenomenon was reported in the Section 4.2 of [21]. To compensate for such effect and

Proposition B.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*

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

where M denotes the memory size.

Proof. It 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

$$\begin{aligned} P_{n,t} &= P_n(n) \cdot P_n(n+1) \cdots P_n(t-1) \cdot P_n(t), \\ &= \frac{M}{n} \cdot \frac{n}{n+1} \cdot \cdots \frac{t-2}{t-1} \cdot \frac{t-1}{t}, \\ &= \frac{M}{t}. \end{aligned} \tag{11}$$

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

C Efficiency Analysis

Due to its simplicity, feature graph requires less learnable parameters. Take the ogbn-arXiv dataset as an example, the number of model parameters including MLP, GCN, SAGE, GAT, APPNP, and FGN used in Section 5 are listed in Table 7. As reported in [56], GCN is the fastest graph learning method, however, FGN requires only 58% parameters compared to GCN, which demonstrates its efficiency.

Table 7: The number of model parameters used in Section 5.

Model	MLP	GCN	SAGE	GAT	APPNP	FGN
# Parameters	37,888	37,888	37,932	38,700	38,184	**21,872**