Meta Continual Learning on Graphs with Experience Replay

Anonymous authors
Paper under double-blind review

Abstract

Continual learning is a machine learning problem where the challenge is that a constructed learning model executes incoming tasks while maintaining its performance over the earlier tasks. In order to solve this problem, we devise a technique that combines two uniquely important concepts in machine learning, namely "replay buffer" and "meta learning", aiming to exploit the best of two worlds. In our approach, first, the model is trained by using the current task dataset, and the model weights for the current task are calculated. Next, we merge the dataset of the current task with the stored samples from the earlier tasks and update the model parameters by training the model with the calculated parameters and the extended dataset. This allows us to distance the parameters from the optimal parameters for the current task and keep the model aligned for the earlier tasks. Furthermore, we choose to adapt our technique to graph data structure and the task of node classification on graphs. We demonstrate that our new method, the MetaCLGraph, outperforms the baseline methods over various graph datasets including Citeseer, Corafull, Arxiv, and Reddit.

1 Introduction

Deep learning models have proven to perform successfully at numerous machine learning tasks including classification and regression. Despite their celebrated performances, deep learning models tend to provide poor results when they are expected to learn from a sequence of data or tasks (Li & Hoiem, 2017). This process is called continual learning, which aims to train a deep learning model such that it manages learning different tasks in order while avoiding forgetting the tasks that it has learned previously. In continual learning, a model is trained in a way such that it can be retrained for future tasks while preserving information from earlier tasks. The main challenge of continual learning is catastrophic forgetting (McCloskey & Cohen, 1989; Goodfellow et al., 2013), which causes the model to lose obtained information from earlier tasks. Catastrophic forgetting is a decisive factor in continual learning research that focuses on the performance drop in the earlier tasks.

The studies addressing the catastrophic forgetting problem in continual learning can be divided into three groups. The first group comprises the memory based studies which either use examples (Rebuffi et al., 2017; Lopez-Paz & Ranzato, 2017) or generate pseudo samples using the stored information (Lavda et al., 2018; Atkinson et al., 2018). The second group focuses on parameter isolation from earlier tasks such that the model can preserve its performance on earlier tasks (Mallya et al., 2018; Serra et al., 2018), while the third group is the regularization-based methods which propose an extra regularization term to preserve information from earlier tasks while learning on a new task (Kirkpatrick et al., 2017; Li & Hoiem, 2017). While the main problem of continual learning is catastrophic forgetting, continual learning is also challenging due to the differences in the problem setup.

Generally, continual learning has two different setups: class incremental and task incremental (De Lange et al., 2021). In class incremental learning, the individual classes are presented sequentially (Masana et al., 2022) while only the tasks are presented in task incremental learning(De Lange et al., 2021). Class incremental setup requires a deep learning model to classify across the observed and current classes whereas the task incremental setup requires an indicator to separate tasks since the model predicts classes within each task in the task incremental setup. Class incremental setup is more challenging than task incremental setup since the model has no indicator of which task is tested (Zhang et al., 2022). In addition, class incremental

Algorithm 1 ER-GNN

for $t = 1$ *to* M **do**
  Calculate loss with T_t
  for $t' = 1$ *to* $t - 1$ **do**
    Get $T_{t'}$ from $\mathcal{B}$
    Calculate loss with $T_{t'}$
  end for
  Sum Losses from T_t & $\mathcal{B}$
  Extend $\mathcal{B}$ with T_t samples
end for

Algorithm 2 LA-MAML

for $t = 1$ *to* M **do**
  Calculate weights for T_t
  Meta loss with calculated weights
  Update learning rates
end for

Algorithm 3 MetaCLGraph

for $t = 1$ *to* M **do**
  $\mathcal{G}_{aux} \leftarrow \mathcal{B} \cup T_t$
  Calculate weights with T_t
  Meta loss with calculated weights on $\mathcal{G}_{aux}$
  Update learning rates
  Extend $\mathcal{B}$ with T_t samples
end for

Figure 1: The algorithms for ER-GNN, LA-MAML, and MetaCLGraph (our approach). $\mathcal{B}$ represents the initially empty replay buffer while M is the number of tasks and T_t is the current task. ER-GNN uses the replay buffer to store examples from tasks while LA-MAML uses meta learning for continual learning. MetaCLGraph fuses both by using meta learning in the training phase while storing examples on the replay buffer.

learning becomes more challenging as the number of classes increases and the data concerning the earlier tasks is not provided.

Continual learning aims to adjust to newly incoming tasks without forgetting the older ones. Meta learning has the potential to provide benefit to the continual learning problem as the aim of the meta learning is to provide a model with the capability to generalize itself to new tasks. Meta learning is used for scarce data regimes (Finn et al., 2017) and typically refers to processes in which a model learns how to learn. For instance, meta learning becomes an important method for few-shot learning in which deep learning models have very few samples to train with (Nguyen et al., 2019; Chen et al., 2023). The meta learning paradigm guides the neural network model weights so that the model weights do not fit the optimal parameters solely for a given task. As parameters that are optimal for each task cannot be attained, the model tends to find the set of parameters that can collectively adapt to all observed tasks. The model parameters are calculated for the current task without any update. After that, the loss which is called the meta loss, is calculated with the earlier calculated weights, and the model is updated with the meta loss. This mitigates catastrophic forgetting as well as quick adaptation to new tasks. The meta learning method can also be applied to graph structured data (Zhou et al., 2019; Tan et al., 2022).

Graph structured data is a type of data that consists of vertices and edges. In addition, graph structured data is non-Euclidean as it does not have a specific hierarchy and order (Asif et al., 2021). Hence, graph structured data does not have any certain geometry. In certain real world applications such as citation networks or online social networks, data that are represented by graphs dynamically change, hence tend to expand continuously. Due to the expansion of the data, two problems may arise: (i) new classes may emerge, and the deep learning model cannot handle the newly arrived classes since the model is not trained for those classes; (ii) on the contrary, if the model is trained with the new classes, they may lose the earlier information. Therefore, continual learning on graph structured data becomes necessary.

In our work, we focus on graph continual learning which addresses continual learning on a graph structure. Our approach fuses the best of two worlds on continual learning for the graph data type, merging meta learning and replay buffer in a single method with learnable learning rates for the first time, to the best of our knowledge. The comparison of our method to the algorithmic families that build on replay buffer, and meta learning is given in Figure 1. Our developed method fuses the selection method in the ER-GNN (Zhou & Cao, 2021) and the meta learning and learnable learning rates from the LA-MAML (Gupta et al., 2020) while integrating those for continual learning of graphs. In the graph continual learning problem, we focus on the task of node classification for the class incremental setup, which requires the model to "remember" earlier classes so that the model can avoid catastrophic forgetting. Our approach has outperformed the benchmark methods on various graph datasets.

2 Background

Graph Neural Networks (GNN) are constructed to process the graph structured data and their inference problems (Kipf & Welling, 2016). Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_i\}_{i=1}^{|\mathcal{V}|}$ denotes the set of nodes and $\mathcal{E} = \{e_{ij}\}_{i,j=1}^{|\mathcal{E}|}$ denotes the edges of the graph, the output embeddings $\mathbf{h}_v^{(l)}$ of a GNN are calculated in two phases. First, the embeddings are obtained by multiplying the weight matrices of each layer $W^{(l)}$ with the embeddings calculated from earlier layers $\mathbf{h}_v^{(l-1)}$. Next, the embeddings of the neighboring nodes to node v are aggregated by using an aggregation function, and the resulting neighboring embedding is further aggregated with the embedding of the node v from the earlier layer. Separating the embedding of node v from the neighboring embeddings allows GNN to preserve the information from node v.

GNNs allow us to consider the interactions between the objects, hence the structure of the adjacency matrix. They also enable the interactions to be reflected onto the features so that the tasks concerning the graph structured data can be completed. GNNs are used for node and graph classification while they are also useful for the link prediction between nodes (Asif et al., 2021; Wu et al., 2022).

Since GNNs are used to classify the nodes by using the connections between them and the graph structured data in real life typically gets modified, the GNNs must be altered in a way such that they can learn the incoming new knowledge while also preserving the information from earlier stages. In order to achieve that, continual learning is applied on GNNs so that the trained GNN model can store earlier information while learning the newly arrived tasks.

One of the common solutions to the continual learning problem is to use a replay buffer that stores examples from earlier tasks (Zhou & Cao, 2021). In this approach, the model must find a way to "remember" the previously learned task because the data stream from earlier tasks is stopped once the model is trained on a task. Using the replay buffer, the model adjusts its weights so that while its weights are trained according to the newly arrived tasks, the model weights would not move away from the earlier tasks.

Meta learning is a method that aims to train a model on a variety of tasks (Finn et al., 2017). Given the task set $T = \{T_1, T_2, \ldots, T_i, \ldots, T_M\}$ where M denotes the number of tasks, the model initially calculates the task i's set of parameters which is denoted as ϕ_i. This process is called the inner update that is used to find the set of parameters for task i. After calculating ϕ_i, the model weights θ are updated by calculating the derivative of ϕ_i with respect to θ. This allows altering the model weights in the direction of the gradient of the loss calculated accordingly to the current task. The objective function for θ is given below where $\mathcal{D}_i^{\text{tr}}$ and $\mathcal{D}_i^{\text{ts}}$ represent the training and test set for task i, respectively.

$$\min_{\theta} \sum_{\text{task } i} \mathcal{L}\left(\theta - \alpha \nabla_{\phi} \mathcal{L}\left(\theta, \mathcal{D}_i^{\text{tr}}\right), \mathcal{D}_i^{\text{ts}}\right). \tag{1}$$

Meta learning is one of the most useful approaches that prevent the weights of a learning model from converging to the optimal parameters of only the current task. The latter is undesirable because the model would only fit to or "remember" a certain task, and would not be able to adapt to newly incoming tasks. We note that this is different from the catastrophic forgetting problem, where the model forgets the earlier tasks. In meta learning, while avoiding convergence to the optimal parameters for any of the incoming tasks, the model targets a "collective intelligence" by converging to a set of parameters that allows the model to perform over all observed tasks.

3 Related Work

Continual Learning mainly deals with the catastrophic forgetting problem. In order to overcome the latter, three approaches are generally used. The first approach is to isolate the parameters so that the previously obtained information would not be lost. Important parameters for the current task are determined, and those parameters are isolated such that they would not change with the future tasks, and the model can still perform for that specific task (Serra et al., 2018; Mallya et al., 2018). The second approach is to use a regularization term such that the weight importance can be considered, and the changes in the

weights that are important for the past tasks are penalized (Kirkpatrick et al., 2017; Aljundi et al., 2018). The regularization prevents the model weights from fitting the optimal parameters of the current task, and hence from forgetting the old tasks. The third approach is to use an additional memory that maintains some examples from the previous tasks. The training process for future tasks uses certain examples from earlier tasks, hence the model would not lose previously obtained information. The usage of additional memory allows the model to observe examples from earlier tasks while it concurrently observes the current task. As a result, the weights would not distance themselves from those of the earlier tasks. The additional memory is used to store information about the earlier tasks to generate similar samples, or it can store examples from the tasks, and those examples can be observed during future tasks. The idea of additional memory is shown to be effective against catastrophic forgetting (Lopez-Paz & Ranzato, 2017; Zhou & Cao, 2021).

Graph Neural Networks recently have been used in continual learning on graphs (Wang et al., 2020). Due to the nature of the graphs, the relationship between the nodes and the knowledge held by the graph is discovered naturally by the use of GNNs since GNNs consider both the adjacency matrix and the features as mentioned earlier. Graph continual learning is a very recent topic (Zhou & Cao, 2021) where GNNs are employed. Since graph structured data obtains objects which are connected to each other, the tasks obtained by dividing the graph provide several connections among different tasks. Hence, learning different tasks can improve the performance of the GNN for earlier tasks, as different connections can be discovered with every incoming task, and this can be used as a remedy against catastrophic forgetting. In real-world scenarios, graphs tend to be dynamic and the boundaries between the tasks do not usually exist (Tang & Matteson, 2020). GNN can find the boundaries between the tasks while being trained task by task on a graph. Graph continual learning is a natural solution in such a scenario. Due to the nature of the real-world graphs, the GNN model becomes more powerful with continual learning, as the same model can be used for different tasks. This allows us to avoid training different models for different tasks, which is in line with the ultimate goal of lifelong learning.

Memory Mechanism. The usage of memory shows to be really effective (Knoblauch et al., 2020) because during the training of the model for a new task, the model sees the early-observed examples, and tends to adjust its parameters so that the performance of the model for earlier tasks can be maintained. Different types of memory units are implemented for the continual learning problem. In one approach, the implemented memory mechanism can store examples from earlier tasks such that when the model is learning future tasks, it is also trained on the examples on the memory (Zhou & Cao, 2021; Lopez-Paz & Ranzato, 2017). This allows the model not to differentiate from the learned parameters of the earlier tasks. Another approach stores some representations from each task such that those representations can be used to generate examples belonging to earlier tasks (Rebuffi et al., 2017). The generated early-task examples are also used during the training of the current task. Hence, the model can preserve its performance while learning the current task with this approach.

Meta Learning is used to train a model on learning to perform multi task learning on a large set of tasks. This technique can be used in different fields such as reinforcement learning and few-shot learning (Finn et al., 2017). Meta learning is used to determine the optimal parameters for different tasks since the main aim of meta learning is to find the set of parameters that can be used for all observed tasks. Once the optimal parameters for a task are determined, the determined parameters are introduced to the model, and the gradients are calculated accordingly so that the original model parameters can be updated with the calculated gradients (Gupta et al., 2020). Meta learning prevents overfitting for a specific task in this way and since the model parameters are updated according to the directions for optimal parameters for each task, the model can find the set of parameters that can be used for every observed task. This approach is effective in mitigating catastrophic forgetting. Meta learning prevents the model weights from fitting a specific task's parameter set causing the model not to overfit to a task and the learning process would not be stopped.

Our proposed approach, namely Meta Continual Learning for Graphs with Experience Replay (MetaCL-Graph), uses meta learning to learn incoming tasks while increasing its efficiency with a memory mechanism. MetaCLGraph focuses on the node classification problem in the class incremental learning setting. It uses

Algorithm 4 MetaCLGraph-Detailed Algorithm

The model weights θ, learning rates α, Graph of current task $\mathcal{G}_t$, Buffer $\mathcal{B}$, number of tasks M, learning rate for α: η

for $t = 1$ *to* M **do**
  Join buffer samples with the current task dataset: $\mathcal{G}' = \mathcal{G}_t \cup \mathcal{B}$
  for $epoch = 1$ *to* E **do**
    Inner update with the $\mathcal{G}_t$: $\theta_{fast} = \theta$ - $\alpha^t \cdot \nabla_\theta \; L_t\,(\theta, \mathcal{G}_t)$
    Learning rate update: $\alpha^{t+1} = \alpha^t$ - $\eta \cdot \nabla_{\alpha^t} \; L_t\,(\theta, \mathcal{G}_t)$
    Meta update with the $\mathcal{G}'$: $\theta = \theta_{fast}$ - $max(0,\, \alpha^t) \cdot \nabla_{\theta_{fast}} \; L_t\,(\theta, \mathcal{G}')$
  end for
  Save samples to buffer $\mathcal{B}$ with coverage maximization
end for

a meta learning mechanism to keep the information from earlier tasks while learning the current task and incorporates a replay buffer as the memory mechanism that stores examples from earlier tasks. Meta learning improves the model's ability to store information from earlier tasks while the memory mechanism allows the model to observe samples from earlier tasks while learning a new one. In addition, learnable parameters are introduced to the model weights as learning rates so that the updates of each parameter can be different.

4 Method

4.1 Problem Setup

The goal is to predict the classes of the nodes for a collection of tasks $T = \{T_1, T_2, \ldots, T_M\}$, where M is the number of tasks. The continual graph problem here entails the model to learn those series of tasks in T. For each task T_i, we have a training node set D_i^{tr} and a test node set D_i^{tst}. Node classification aims to predict the right class for each node, i.e., to classify each node in the test node set D_i^{tst} into the correct class by learning the tasks using D_i^{tr}. In our graph continual learning setup, we aim to classify incoming nodes based on early observed classes which is also known as class incremental learning (Masana et al., 2022).

4.2 Our Method: MetaCLGraph

We devise a new method for graph continual learning named MetaCLGraph. We introduce a meta learning solution to continual node classification by incorporating an episodic memory for experience replay. The algorithm for our method is given in Algorithm 4. MetaCLGraph leverages the strengths of both experience replay through a memory buffer (Zhou & Cao, 2021) and meta learning (Gupta et al., 2020) to mitigate the catastrophic forgetting problem. During the training, the model weights for the current task are calculated before updating the actual weights. After the calculation of the model weights for the current task, the model is trained with the calculated weights using the data obtained by using the experience replay & the incoming data, where the original model weights are updated regarding the updates on the earlier calculated fast weights. This is called the meta update and it allows our model not to fit the optimal parameters for a given task. The experience replay is obtained when the model is trained on a task and some nodes are selected by using the coverage maximization function (4.2.1). The replay buffer can also be seen as an episodic memory since the buffer stores the nodes from the current task at the end of the training. The replay buffer is constructed in this way so that the finished tasks can be revisited by the model using the buffer. The main reason for using the meta learning with a buffer is to prevent the model weights from positioning close to the optimal parameters for a certain task. Following a continual learning paradigm, MetaCLGraph adjusts the model weights to the incoming tasks. Revisiting the stored sample nodes allows the model to remember the earlier tasks. With this added reminder process, the model avoids adapting to only one task. It is able to distance its parameters from the optimal parameters of the current task and also stays tuned to the earlier tasks.

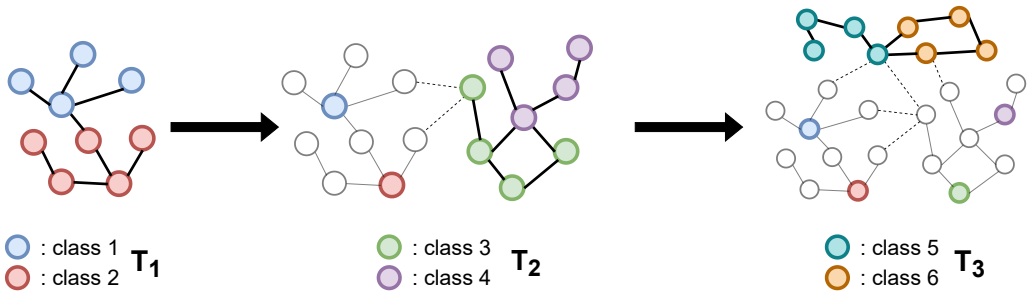

Figure 2: The dataset merging process is illustrated in this figure. The arrows represent the transition between the tasks while coloured nodes represent the active nodes during that task. In our approach, multiple nodes are selected at the end of the task and stored inside the replay buffer. When a new task is arrived, stored nodes are merged with the current task and form a merged dataset.

4.2.1 Experience Replay

At the end of the training for each task, the additional memory is updated by selecting some examples from that particular task. The additional memory allows our model to "remember" previous tasks as the model weights are trained for the next task. As the model also observes the selected examples, the experience replay allows our model to preserve knowledge from earlier tasks. However, the selection process for the experience replay is important as the representation of a given task relies on the stored nodes.

During the sample selection, we use the coverage maximization method (Zhou & Cao, 2021) in order to determine the nodes which can represent their tasks accordingly. Coverage maximization is used to find the distance of nodes to the other classes. A node with a certain class can represent its class by being more distant from the other classes. The distances for each node are calculated and the nodes with the shortest distance to the nodes from other classes are selected in order to represent their classes. These nodes are used for the merge operation explained in Section 2.

The experience replay allows us to construct a merged dataset as illustrated in Figure 2. When the training process for a task is finished, several nodes are selected according to the coverage maximization function and stored inside the experience replay. After that, when a new task arrives, the nodes stored inside the experience replay are merged with the data of the current task, and a merged graph is stored. The merged graph allows us to train our model accurately since the model can keep the information learned earlier while learning the current task.

4.2.2 Meta Learning

In order to preserve the continual learning ability of MetaCLGraph, the meta learning is adapted to our graph neural network as follows. When learning a task T_i, suppose that the training node set D_i^{tr} arrives. Using the buffer, the model merges the D_i^{tr} with the stored sample nodes from earlier tasks. The newly obtained training set becomes D_i^{tr+b} as it now contains both the training node set for the task T_i and the buffer samples. After constructing the dataset, the model moves on to the training process.

The model is trained on the merged dataset according to the meta learning setup. First, the model is trained with D_i^{tr} in order to determine the parameters for the current task T_i. The parameters for the current task are determined in order to find the direction of the optimal parameters for that task. In order to achieve that, the model weights are calculated and stored. After that, the gradients are calculated and the model weights, which are called as fast weights θ_{fast}, are updated, shown in Algorithm 4, line 4. The weights are calculated during the training of the current task, as the model is optimized according to the current task. Meanwhile, the model also observes examples from earlier tasks. After the calculation, the model is trained

Table 1: Details of the datasets used in the experiments.

Dataset	# nodes	# edges	# features	# classes	# tasks
CiteSeer-CL	3327	9228	3703	6	3
Corafull-CL	19793	130622	8710	70	35
Arxiv-CL	169343	1166243	128	40	20
Reddit-CL	227853	114615892	602	40	20

when θ_{fast} are introduced to the model, and the loss is calculated. Finally, the initial model weights θ are updated with the loss obtained by using θ_{fast}, shown in Algorithm 4, line 6.

4.2.3 Learning Rate Update

In addition to meta learning and experience replay setups, the MetaCLGraph assigns learning rates to every weight, and the assigned learning rates are also updated during the training. Every weight having a different importance for each task during the training as the model learns a task helps in the preservation of the learned knowledge (Gupta et al., 2020).

Using the meta learning loss calculated for the merged dataset, the learning rates for each parameter are also updated by using the gradients. With the updates on the learning rates, the updates on our model weights become more foreseeable. If the model weights require more updates in order to learn the task better, then the learning rate update would be limited. However, if some of the model weights have reached an optimal region, then their learning rate decreases more dramatically than the learning rate of the other weights since the model would tend to preserve that information. This update difference among the model weights allows our model to adjust the parameters according to the tasks, hence the already-acquired information is not lost.

5 Experiments

The continual learning experimental setups for the GNNs are investigated, and the performance of our setup is compared with the other continual learning setups. The pipeline provided by (Zhang et al., 2022) is used for the experiments. The datasets, baselines, experimental settings, and results are reported and described next [1].

5.1 Datasets

To evaluate our model MetaCLGraph, four benchmark datasets were used: Corafull (Bojchevski & Günnemann, 2017), Arxiv (Hu et al., 2021), Reddit (Hamilton et al., 2017), and Citeseer (Sen et al., 2008). Arxiv, Corafull, and Reddit are used in (Zhang et al., 2022) to construct a benchmark for continual graph learning. Meanwhile, Citeseer data is an important graph type data for evaluation, too. In this problem, we divide the datasets into tasks containing two classes for each task. For the Corafull dataset, 35 tasks are obtained; for the Citeseer dataset 3 tasks are obtained, while Arxiv and Reddit datasets contain 20 tasks. For the Reddit dataset, the 41st and the last class is dropped since it only has one example. Table 1 provides the detailed task information, and graph structures for each dataset.

For larger graphs, it is necessary to split them into batches due to the device memory requirements. Reddit dataset is the largest graph amongst our datasets, therefore it is divided into batches. Each of the other three datasets is taken as a whole batch.

5.2 Baselines

The following baselines are used in order to evaluate the MetaCLGraph.

[1] The source codes will be provided during publication to ensure reproducibility.

Base model is the graph neural network without using any continual learning components. The data is simply passed to the GNN architecture and the model is trained with the provided data. There are no further improvements on the GNN architecture.

Elastic weight consolidation (EWC) (Kirkpatrick et al., 2017) is a method that determines the importance of model weights for each task and preserves the weights according to their importance level while weights with lower importance are adjusted for the incoming tasks. EWC allows the model to learn the important parameters for the current task and when future tasks arrive, the model preserves the important parameters for the earlier task and adjusts its other parameters so that it can preserve learned knowledge.

Memory aware synapses (MAS) (Aljundi et al., 2018) is a regularization based method that evaluates the importance of the parameters according to the sensitivity of the predictions on the parameters. When new data arrives, the model weights are updated according to the activations of the network so that the important model weights for the newly acquired data can be updated.

Topology-aware weight preserving (TWP) (Liu et al., 2021) is a method introduced for graph continual learning which preserves weights regarding the topology of the provided graph. Graph structure is taken into consideration in order to prevent catastrophic forgetting. After the loss is calculated, it is regularized by calculating the importance score of the model weights according to the topological structure of the provided graph. It benefits from the properties of the graph.

ER-GNN (Zhou & Cao, 2021) uses experience replay for this problem. After learning a task T_i, sample nodes are saved to the buffer with a selection function and when T_{i+1} is being learned, separate graphs for each learned task from k={0 to i} are constructed, and the GNN is trained with those graphs. The overall loss is calculated with the loss for the current task regularized by the loss calculated from the constructed separate graphs using the experience replay.

MetaCLGraph (Ours) uses an episodic memory buffer, and merges meta learning with experience replay.

Joint is the method where all the learned tasks are accessed during training. Therefore, it sets the upper bound for our setup.

EWC and MAS are the techniques focusing on parameter isolation, while ER-GNN uses the additional memory as a replay buffer to store examples from earlier tasks, however, lacking any meta-learning component. TWP is a regularization method that considers the topology of the graph while updating weights. Our model, the MetaCLGraph is compared with those baseline techniques for performance assessment.

5.3 Experiment Settings and Performance Scores

The experiments are conducted with a learning rate of 0.005, and each task is trained for 200 epochs. The batch size is selected as 2000 for the batched datasets. Adam optimizer (Kingma & Ba, 2014) is selected as the optimizer. All methods use the graph convolutional network as the backbone GNN architecture (Kipf & Welling, 2016). The selection algorithm relies on coverage maximization. The hyperparameters concerning the compared methods are obtained from the benchmark paper (Zhang et al., 2022) and its repository, as the results in the benchmark are reproducible. The buffer budget is selected as 10 for the replay based methods, namely the ER-GNN and the MetaCLGraph. Given in the benchmark paper (Zhang et al., 2022), the buffer budget sets the number of samples selected for each class in the task, and the entire class or task can be stored when the buffer budget is set high. Setting the buffer budget high does not serve the purpose of continual learning since it would become likely that an entire class or task can be observed in future tasks. To avoid storing an entire class or task that would defeat the purpose of continual learning, the buffer budget is determined as 10. This is based on the observation that all of the datasets contain more than 10 samples for each class. Hence, the buffer budget becomes strict where storing an entire class or task is avoided.

The two evaluation measures are the Average Performance (AP) and the Average Forgetting (AF) (Lopez-Paz & Ranzato, 2017). AP focuses on the model's accuracy in classification of each task. It is obtained

Table 2: The AP and AF measure results across four datasets for various baselines and the MetaCLGraph. The results are showing mean ± standard deviation across 5 repetitions. Best performances are highlighted in bold.

Method	Citeseer		CoraFull		Arxiv		Reddit	
	AP ↑	AF ↑	AP ↑	AF ↑	AP ↑	AF ↑	AP ↑	AF ↑
Base Method	31.49±0.15	-77.48±0.29	2.24±0.20	-94.82±0.33	4.85±0.04	-87.17±1.97	5.49±1.05	-93.39±1.28
EWC	31.42±0.15	-77.46±0.32	15.02±2.14	-81.67±2.32	4.86±0.01	-88.35±0.45	8.88±1.64	-94.56±1.75
TWP	31.37±0.08	-78.43±0.26	16.21±1.76	-77.81±1.57	4.86±0.02	-88.89±0.13	11.75±1.60	-91.59±1.81
MAS	31.58±0.07	-77.47±0.32	6.85±1.89	-88.52±2.14	4.86±0.06	-86.52±0.62	12.58±4.67	**-26.24±8.53**
ER-GNN	46.81±0.33	-51.10±0.45	2.26±0.32	-95.12±0.40	27.83±0.26	-54.67±0.17	37.64±0.64	-64.55±0.64
MetaCLGraph (Ours)	**51.48±1.64**	**-46.79±2.78**	**64.73±0.34**	**-16.86±0.38**	**31.76±0.74**	**-51.03±0.82**	**67.66±3.64**	-33.01±3.86
Joint (Upper bound)	76.40±0.20	-	81.56±0.14	-	46.35±0.85	-	98.33±0.26	-

by calculating the mean accuracies of the observed tasks so far. AF considers whether or not the model preserves its performance on the observed tasks. The AF is calculated as follows:

$$a_{j,j} - a_{k,j}, \quad \forall j < k \tag{2}$$

where $a_{j,j}$, and $a_{j,j}$ represent the performance scores obtained from the accuracy matrix. j and k represent the numbers of tasks where k represents the current task and j represents a certain old task when it was trained as the current task. In summary, forgetting is determined by finding the performance difference between the last performance and the performance obtained as the current task for each task. All experiments are repeated 5 times on one Nvidia RTX A4000 GPU.

5.4 Results

The experimental results are reported in Table 2. The results indicate that the proposed MetaCLGraph outperforms the compared baselines, and achieves the best performance across all datasets in terms the AP measure, and the best performance in terms of the AF measure excluding only the Reddit dataset. These results provide evidence for the fact that the proposed method allows the model to learn the tasks and stores information about the early-observed tasks. The MetaCLGraph outperforms the regularization based methods such as MAS, TWP, and EWC by employing its replay buffer. As mentioned earlier, regularization based methods alter the loss such that the model can preserve the obtained information. However, our proposed method uses the stored examples during the learning of the future tasks as those examples are observed during their training. Therefore, the utilization of the replay buffer allows the MetaCLGraph to outperform regularization based methods. Although MAS has a better AF score for the Reddit dataset, this score is obtained since the AP score is low. Therefore, it can be deduced that the MAS method cannot learn the tasks for the Reddit dataset, and its AF score does not mean a better performance as the MAS method has not learned enough information about the observed tasks for storage. The MetaCLGraph outperforms ER-GNN which also utilizes a replay buffer. Although they use the same selection function for the examples to be stored inside the replay buffer, the MetaCLGraph exploits meta learning during the training process. During training, the optimal model parameters are determined for the current task, however, the loss is not calculated until the model observes the stored examples from the earlier tasks. Therefore, the model does not exhibit over-fitting to the current task. On the contrary, it finds the set of parameters that can perform for the earlier tasks while learning the current task. The calculation of the meta loss allows the model to adjust the parameters considering the earlier tasks, hence leads to MetaCLGraph model outperforming the ER-GNN method.

In Figure 3, the performance matrices of the benchmark methods and MetaCLGraph are visualized. The performance scores are presented for all of the tasks after the observation. It can be observed for each dataset that the changes in the performance scores for MetaCLGraph are less pronounced than those for other methods, thanks to a relatively better preservation of the learned information in the former.

5.5 Ablation

We investigate how meta learning and experience replay affect the MetaCLGraph's performance on class incremental learning. In this part, we compare the performances with the versions of our model, which do

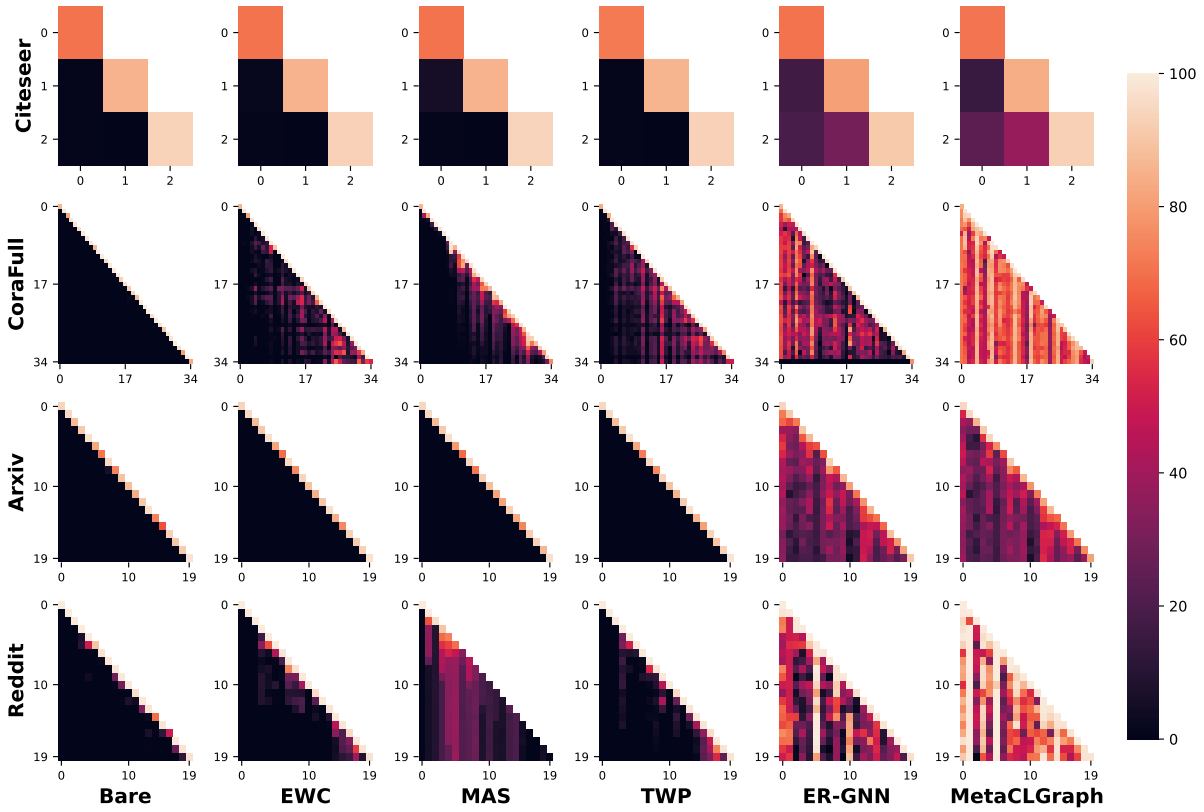

Figure 3: The visualization of performance matrices for MetaCLGraph and benchmark methods for the trained datasets. Each entry in the matrices represents the performance of the method on the column task while learning the task on the row. The determined performance metric on the visualization is accuracy where lighter colours represent the higher score while darker colours represent the lower scores.

not involve experience replay or meta learning. For testing the effects of experience replay, the buffer budget for the experience replay is changed. When the buffer budget is 0, this would be our approach without experience replay. On the other hand, our method without meta learning would be equivalent to ER-GNN whose results were given in Section 5.4.

The results for the ablation study are given in Figure 4. It can be observed that the model only using meta learning gives similar results to those of the base model. Since this version of our proposed model does not store any information from earlier tasks, and only relies on meta learning, it only learns the current task. Therefore, its AP and AF scores would be similar to the base model since it adjusts its parameters on only the current task and not the other tasks. It is also observed that the number of stored samples increases our model's performance significantly. The number of samples for each class is determined as 10 in order to avoid storing an entire class or task inside the buffer. The results show that when the capacity of the buffer is increased, the performance is improved. However, increasing the capacity too much may cause storing tasks, and storing the tasks would not serve the purpose of continual learning. Although some examples were stored inside the buffer, continual learning setup requires the continuous data flow from earlier tasks to be stopped once the training for a task is finished since the finished tasks become unreachable after the training. However, when the tasks start to be stored in the replay buffer, the model will be retrained on the already visited tasks. Therefore, the continual learning setup is no longer valid, and the model would be trained in a traditional supervised learning setup. This is observed in average forgetting as well. Since all examples from a class may have been stored with a high buffer budget, the model is trained on a task more than once. Therefore, the average forgetting rate becomes greater than zero due to the fact that it has been

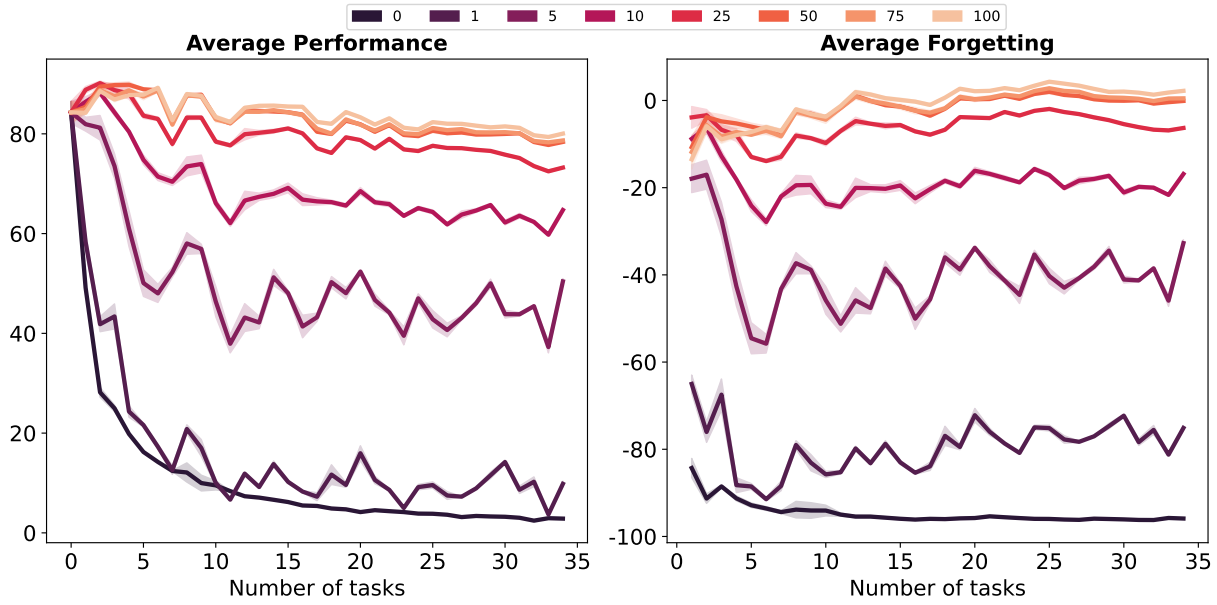

Figure 4: The effect of buffer budget on MetaCLGraph is shown for AP (left) and AF (right). As the buffer budget increases, both the model performance and forgetfulness are improved due to the increase in the stored number of samples per class.

trained more than specified epochs and the model's performance on that task is improved by meta learning and additional epochs. For the purposes of continual learning, this is avoided and the buffer has a limit on stored examples.

6 Conclusion

Summary. We present the MetaCLGraph for graph continual learning which merges the meta learning paradigm with the use of a replay buffer. MetaCLGraph is compared with the baseline continual learning methods and graph continual learning methods. The experiments provide evidence to our hypothesis that the meta learning paradigm improves the efficiency of the replay buffer and mitigates the catastrophic forgetting problem by conserving the obtained information from earlier tasks. It is observed that the MetaCLGraph outperforms the corresponding baselines in terms of average performance and average forgetting measures.

Broad Impact. MetaCLGraph framework can be used in applications for expanding networks where new members and connections are emerging and their characteristics are discovered. The replay buffer of our model can be altered with a different type of memory that store the class representations instead of class examples in order to address data security concerns.

Future Work. Our work has shown that the utilization of meta learning improves the efficiency of using a memory mechanism such as a replay buffer. Expanding from MetaCLGraph, a future research direction could be changing the memory mechanism from a replay buffer to a subconscious memory that stores representations for every observed class.

Limitations. Storing examples from earlier tasks creates a dilemma for the memory based solutions in terms of data sharing. Our method performs storing a limited number of examples, hence, limiting the buffer capacity and not storing the entire incoming task or class. By limiting the number of examples stored in the replay buffer, our method serves well for the purposes of continual learning while alleviating data privacy and ethical concerns.

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.

Nurul A. Asif, Yeahia Sarker, Ripon K. Chakrabortty, Michael J. Ryan, Md. Hafiz Ahamed, Dip K. Saha, Faisal R. Badal, Sajal K. Das, Md. Firoz Ali, Sumaya I. Moyeen, Md. Robiul Islam, and Zinat Tasneem. Graph neural network: A comprehensive review on non-euclidean space. *IEEE Access*, 9:60588–60606, 2021. doi: 10.1109/ACCESS.2021.3071274.

Craig Atkinson, Brendan McCane, Lech Szymanski, and Anthony Robins. Pseudo-recursal: Solving the catastrophic forgetting problem in deep neural networks. *arXiv preprint arXiv:1802.03875*, 2018.

Aleksandar Bojchevski and Stephan Günnemann. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. *arXiv preprint arXiv:1707.03815*, 2017.

Lisha Chen, Sharu Theresa Jose, Ivana Nikoloska, Sangwoo Park, Tianyi Chen, Osvaldo Simeone, et al. Learning with limited samples: Meta-learning and applications to communication systems. *Foundations and Trends® in Signal Processing*, 17(2):79–208, 2023.

Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Aleš Leonardis, Gregory Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE transactions on pattern analysis and machine intelligence*, 44(7):3366–3385, 2021.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pp. 1126–1135. PMLR, 2017.

Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013.

Gunshi Gupta, Karmesh Yadav, and Liam Paull. Look-ahead meta learning for continual learning. *Advances in Neural Information Processing Systems*, 33:11588–11598, 2020.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs, 2021.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

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.

Jeremias Knoblauch, Hisham Husain, and Tom Diethe. Optimal continual learning has perfect memory and is np-hard. In *International Conference on Machine Learning*, pp. 5327–5337. PMLR, 2020.

Frantzeska Lavda, Jason Ramapuram, Magda Gregorova, and Alexandros Kalousis. Continual classification learning using generative models. *arXiv preprint arXiv:1810.10612*, 2018.

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

Huihui Liu, Yiding Yang, and Xinchao Wang. Overcoming catastrophic forgetting in graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pp. 8653–8661, 2021.

David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. *Advances in neural information processing systems*, 30, 2017.

Arun Mallya, Dillon Davis, and Svetlana Lazebnik. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 67–82, 2018.

Marc Masana, Xialei Liu, Bartłomiej Twardowski, Mikel Menta, Andrew D Bagdanov, and Joost Van De Weijer. Class-incremental learning: survey and performance evaluation on image classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(5):5513–5533, 2022.

Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pp. 109–165. Elsevier, 1989.

Binh D Nguyen, Thanh-Toan Do, Binh X Nguyen, Tuong Do, Erman Tjiputra, and Quang D Tran. Overcoming data limitation in medical visual question answering. In *Medical Image Computing and Computer Assisted Intervention–MICCAI 2019: 22nd International Conference, Shenzhen, China, October 13–17, 2019, Proceedings, Part IV 22*, pp. 522–530. Springer, 2019.

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.

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.

Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. In *International conference on machine learning*, pp. 4548–4557. PMLR, 2018.

Zhen Tan, Kaize Ding, Ruocheng Guo, and Huan Liu. Graph few-shot class-incremental learning. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, pp. 987–996, 2022.

Binh Tang and David S Matteson. Graph-based continual learning. *arXiv preprint arXiv:2007.04813*, 2020.

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*, pp. 1515–1524, 2020.

Lingfei Wu, Peng Cui, Jian Pei, and Liang Zhao. *Graph Neural Networks: Foundations, Frontiers, and Applications*. Springer Singapore, Singapore, 2022.

Xikun Zhang, Dongjin Song, and Dacheng Tao. Cglb: Benchmark tasks for continual graph learning. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022.

Fan Zhou and Chengtai Cao. Overcoming catastrophic forgetting in graph neural networks with experience replay. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 4714–4722, 2021.

Fan Zhou, Chengtai Cao, Kunpeng Zhang, Goce Trajcevski, Ting Zhong, and Ji Geng. Meta-gnn: On few-shot node classification in graph meta-learning. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pp. 2357–2360, 2019.