ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

A MEMORY AND META LEARNING BASED SOLUTION IN GRAPH CONTINUAL LEARNING

M.Sc. THESIS

Altay ÜNAL

Department of Computer Engineering

Computer Engineering Programme

JUNE 2024

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

A MEMORY AND META LEARNING BASED SOLUTION IN GRAPH CONTINUAL LEARNING

M.Sc. THESIS

Altay ÜNAL
(504201566)

Department of Computer Engineering

Computer Engineering Programme

Thesis Advisor: Prof. Dr. Gözde ÜNAL

JUNE 2024

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

ÇİZGELERDEKİ SÜREKLİ ÖĞRENME PROBLEMİNDE HAFIZA VE META ÖĞRENME TEMELLİ ÇÖZÜM

YÜKSEK LİSANS TEZİ

Altay ÜNAL
(504201566)

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Programı

Tez Danışmanı: Prof. Dr. Gözde ÜNAL

HAZİRAN 2024

Altay ÜNAL, a M.Sc. student of ITU Graduate School student ID 504201566 successfully defended the thesis entitled "A MEMORY AND META LEARNING BASED SOLUTION IN GRAPH CONTINUAL LEARNING", which he/she prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Prof. Dr. Gözde ÜNAL** Istanbul Technical University	.....
Jury Members :	**Assoc. Prof. Dr. Melih KANDEMİR** University of Southern Denmark	.....
	Assoc. Prof. Dr. Yusuf YASLAN Istanbul Technical University	.....
		.....

Date of Submission : **22 May 2024**
Date of Defense : **12 June 2024**

To my family and friends,

FOREWORD

First, I would like to thank my advisor Prof. Gözde ÜNAL for her support and guidance throughout this thesis. Her advices helped me in my Master's journey and allowed me to improve myself. I also would like to thank Assoc. Prof. Melih Kandemir for sharing his insights and ideas throughout my research.

Secondly, I would like to thank my family for their endless support through this process. Their support and belief helped me to get through the hardest times.

I especially thank Abdullah AKGÜL for his friendship and support. During this process, we have been through a lot and I am very proud to be working with him. I also met brilliant people through my time at ITU Vision Lab and I thank them for their insights, help, and friendship: Gülçin BAYKAL CAN, Batuhan CENGİZ, Fırat ÖNCEL, Yusuf H. ŞAHİN. I also worked with intelligent and talented undergraduate students: Talha Sezer ÇAKIR and Emircan GÜNDOĞDU. Being their mentor was an enriching experience for me.

Last but not least, I would like to thank my friends: Leman Dicle BALCI, Sarp ÇAVUŞLAR, Canberk KOCAOĞLU, Yiğit Deniz LOKUMCU. I thank them for their friendship.

June 2024

Altay ÜNAL
Research Assistant

TABLE OF CONTENTS

ABBREVIATIONS

AF	**:** Average Performance
AP	**:** Average Forgetting
CL	**:** Continual Learning
ER	**:** Experience Replay
EWC	**:** Elastic Weight Consolidation
GNN	**:** Graph Neural Network
GCN	**:** Graph Convolutional Network
GCL	**:** Graph Continual Learning
LwF	**:** Learning without Forgetting
MAML	**:** Model-Agnostic Meta Learning
MAS	**:** Memory Aware Synapses
MetaCLGraph	**:** Meta Continual Learning for Graphs with Experience Replay
TWP	**:** Topology-aware Weight Preserving

SYMBOLS

θ	: Model weights
θ_{fast}	: Fast model weights
ϕ_i	: The model parameters for task i
$\mathscr{G}$	: Graph
$\mathscr{G}_t$	: Graph of current task
$\mathscr{B}$	: Buffer
α	: Learnable learning rates
η	: Learning rate for learnable learning rates
L_t	: Loss function for task t
∇	: Backpropagation
$\mathscr{V}$	: Vertices
$\mathscr{E}$	: Edges
θ	: Model weights
T_i	: i^{th} task
$a_{i,j}$	: Performance score for the i^{th} task while j^{th} task is current task
$\mathscr{N}$	: The set of nodes
d	: Distance
$\mathscr{Y}(v_i)$	: The label of node v_i
D_i^{tr}	: The training dataset for the task i
D_i^{tst}	: The test dataset for the task i

LIST OF TABLES

LIST OF FIGURES

A MEMORY AND META LEARNING BASED SOLUTION IN GRAPH CONTINUAL LEARNING

SUMMARY

Deep learning models have proven to perform successfully at different tasks such as classification and regression. Continual learning (CL) aims for a model to learn various tasks sequentially. However, when the models are expected to adapt to incoming tasks without maintaining their performance on previous tasks, they tend to forget the previous tasks. This phenomenon is called catastrophic forgetting and catastrophic forgetting is the main challenge in the CL area. Catastrophic forgetting refers to the scenario where a model tends to forget the previous tasks it had been trained on and adjusts its parameters to perform the task it is actively being trained on. Since it is inefficient to train multiple models to perform multiple tasks, CL aims to train a single model such that it can perform on multiple tasks without losing information during the training process. In addition to catastrophic forgetting, CL also focuses on capacity saturation which is another challenge focusing on the effects of the model architecture on learning.

CL is currently an emerging research field topic. However, the CL studies mainly focus on image data and there is much to discover in CL research focusing on graph-structured data or graph continual learning (GCL). The proposed solutions for GCL are mainly adapted from the general CL solutions, therefore, there is much to discover in GCL field. However, since the graph-structured data has different properties compared to image data, the graph properties need to be considered when GCL is studied.

In this thesis, we focus on continual learning on graphs. 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 to successfully achieve continual learning on graph structured data. In this method, the model weights are initially computed by using the current task dataset. Next, the dataset of the current task is merged with the stored samples from the earlier tasks, and the model weights are updated using the combined dataset. This aids in preventing the model weights converging to the optimal parameters of the current task and enables the preservation of information from earlier tasks.

We choose to adapt our technique to graph data structure and the task of node classification on graphs and introduce our method, MetaCLGraph. Experimental results show that MetaCLGraph shows better performance compared to both baseline CL methods and developed GCL techniques. The experiments were conducted on various graph datasets including Citeseer, Corafull, Arxiv, and Reddit.

ÇİZGELERDEKİ SÜREKLİ ÖĞRENME PROBLEMİNDE HAFIZA VE META ÖĞRENME TEMELLİ ÇÖZÜM

ÖZET

Regresyon ve sınıflandırma gibi çeşitli işlerde derin öğrenme modellerinin başarılı şekilde performans gösterdiği ispatlanmıştır. Ancak yeni bir iş geldiğinde daha önceden öğrendikleri işlerde gösterdikleri performansı koruyarak bu işi öğrenmeleri beklenirken bu modeller önceden öğrenilen işleri unutma eğilimindedirler. Bu fenomene yıkıcı unutma denir ve bu fenomen sürekli öğrenme alanındaki ana zorluktur.

Yıkıcı unutma durumu aşağıdaki senaryoda açıklanmaktadır. Bir derin öğrenme modeli bir takım işlerde eğitilmiş ve iyi performans göstermektedir. Sıradaki iş için eğitilirken önceden öğrendiği işleri "unutur" ve bu işlerde performans kaybı yaşanır. Önceki işlerde performans kaybı yaşanırken model bu esnada kendi parametrelerini aktif olarak öğrendiği işte performans gösterecek şekilde ayarlar. Sürekli öğrenmede amaç bu senaryonun gerçekleşmesini önleyerek bir modeli birden fazla işte performans gösterecek şekilde eğitmektir. Birden fazla iş için birden fazla model eğitmek oldukça verimsizdir çünkü hem derin öğrenme modellerinin eğitimi uzun vakit alır hem de bu modelleri barındırmak çok ciddi cihaz hafızası gerektirir. Dolayısıyla sürekli öğrenme bu sorunları çözmek üzere geliştirilmiş bir konsepttir. Yıkıcı unutmanın yanı sıra kapasite doyumunda sürekli öğrenmede odaklanılan başka bir zorluktur. Kapasite doyumunda derin öğrenme modellerinin mimarisi incelenir ve model mimarisinin birden fazla iş öğrenimindeki etkisi araştırılır. Kapasitesi belli olan bir modelin performansı gelen iş sayısı arttıkça sabitlenecek ve model parametreleri doymuş olacaktır. Bu durum sağlamlık esneklik ikilemine yol açmaktadır, bir model yeterince esnek olursa gelen her işi öğrenebilir ancak sağlam olmadığı için önceden öğrenilen işlerle alakalı bilgilerini yitirerek sürekli öğrenmeyi yerine getiremeyecektir.

Sürekli öğrenme araştırmalarında genel olarak üç çeşit yaklaşım ön plana çıkmaktadır. Bunlar parametre izolasyon temelli metotlar, arabellek temelli metotlar ve regülarizasyon temelli metotlardır. Parametre izolasyonu temelli metotlarda işlerde özel olarak ön plana çıkan parametreler belirlenerek bu parametrelerin bir sonraki işlerde güncellenmesi önlenir, bu sayede öğrenilen bilgilerin kaybolmaması amaçlanır. Regülarizasyon temelli metotlarda yitim fonksiyonuna eklenen bir terim ile modelin unutmasının önüne geçilmesi hedeflenirken arabellek temelli metotlarda amaç modele ek olarak bir arabellek tanımlanarak bu arabelleğin modelin eğitiminde rol oynayarak modelin önceki işleri unutmasının önüne geçmesini sağlamaktır.

Sürekli öğrenmede amaç önceki işlere ilişkin bilgiyi korumak olduğu için amaç bir manada model parametrelerinin o işler için optimal olan parametrelerinden uzaklaştırmadan yeni bir işi öğretmektir. Bu noktada da meta öğrenme konsepti

sürekli öğrenme ile kesişmektedir. Meta öğrenmede amaç modelin gelen işleri genelleştirmesini sağlamaktır, modele gelen işi nasıl öğreneceği öğretilir. Böylelikle model gelen işleri uygun şekilde genelleştirerek hepsinde performans gösterebilecek şekilde parametrelerini uyarlar. Meta öğrenmenin bu yönü sürekli öğrenme ile benzerlikler taşıdığı için sürekli öğrenmeye katkı sağlayabilir.

Sürekli öğrenme an itibariyle yükselişte olan bir araştırma konusudur çünkü derin öğrenme modellerinin birden fazla işte performans göstermeleri sorunsuz şekilde gündelik uygulamalara uyarlanmak istenmektedir. Ancak sürekli öğrenme çalışmaları genel olarak görsel veri üzerinden ilerlemektedir. Dolayısıyla çizge verisi üzerine odaklanan sürekli öğrenme ya da sürekli çizge öğrenmesi önü oldukça açık bir araştırma alanıdır. Bununla beraber çizge verisinin görsel veriye göre farklı özellikleri olduğundan bu özellikler de değerlendirmeye alınmalıdır.

Çizge verisi Öklid dışı geometriye sahip bir veridir. Dolayısıyla düzlemsel olarak değerlendirilemez. Çizge verisinde düğümler ve bu düğümleri birbirine bağlayan kenarlar bulunmaktadır. Çizge verisinin derin öğrenme modelleri tarafından öğrenilmesi sırasında düğüm özellikleri dikkate alınır. Buna ek olarak düğümleri birbirine bağlayan kenarlar da değerlendirmeye alınır. Çizge verisinde düğümler birbirleri ile olan ilişkilerine göre kenarlarla bağlandığı için derin öğrenme modelinin bu ilişkileri keşfetmesi gerekmektedir. Aksi takdirde çizge verisi ve o veriyi oluşturan düğümlerin ilişkisi anlaşılamaz, bu da çizge verisinin öğrenilememesine yol açar.

Çizge verisinde sürekli öğrenme ya da sürekli çizge öğrenmesi keşfedilecek pek çok yönü olan bir araştırma alanıdır. Çizge verisinin yaygın hale gelmesiyle beraber çizge verisi üzerinde makine öğrenmesinin yaygınlaşmasına ek olarak çizge verisinin sürekli güncellenen bir veri tipi olması sürekli çizge öğrenmesini yükselişte olan bir araştırma alanı haline getirmiştir. Alıntı ağları veya sosyal ağlar göz önüne alındığında sürekli çizge öğrenmesinin önemi görülecektir. Bu gibi ağlar çizge olarak ifade edilmektedir ve bu ağlarda günümüzde sıklıkla yeni sınıflar oluşmaktadır. Dolayısıyla çizge verisi üzerinde eğitilen modellerin ortaya çıkan yeni sınıfları öğrenmesi gerekmektedir. Aksi takdirde bu modellerin tekrar eğitilmesi gerekmektedir.

Bu teze konu olan çalışmada sürekli çizge öğrenmesi alanına odaklanılmıştır ve "tekrar arabelleği" ile "meta öğrenme" konseptleri kullanılarak sürekli çizge öğrenmesini yerine getiren bir model eğitilmiştir. Bu konseptlerin kullanılma amacı sürekli çizge öğrenmesini yerine getirebilmek için hafıza temelli ve parametre temelli iki konseptin en iyi yanlarının birleştirilmesidir. Söz konusu modelin eğitimi aşağıdaki şekildedir. Öncelikle aktif iş verisi üzerinde model parametreleri hesaplanır. Ardından, önceki işlerden arabellekte depolanan örneklerle aktif işin verisi birleştirilir ve model parametreleri bu birleşik veri kullanılarak güncellenir. Bu sayede model parameterelerinin aktif iş için optimal olan parametrelere yakınsaması önlenirken aynı zamanda önceki işlerin öğrenimi sırasında edinilen bilgilerin korunması sağlanır.

Yukarıda açıklanan şekilde eğitilen bu modelin yapılan deneylerde temel olarak alınan sürekli öğrenme modelleri ve sürekli çizge öğrenmesi modellerine göre daha iyi performans gösterdiği deneysel olarak gösterilmiştir. Deneylerde Citeseer, Corafull, Arxiv ve Reddit gibi yaygın çizge verileri kullanılmıştır. Bu deneylere ek olarak hesaplama bedeli deneyi ile arabellek ve meta öğrenmeye odaklanan ablasyon deneyleri de yapılarak kullanılan konseptlerin modelin eğitiminde olan

etkileri gösterilmiştir. Bu deneyler sonucunda hem meta öğrenmenin hem tekrar arabelleğinin modelin performansına yaptığı katkılar görülmüştür. Buna ek olarak tekrar arabelleğinde tutulan örnek sayısının etkileri de araştırılmış ve sürekli öğrenmeye olan etkileri de görülmüştür.

1. INTRODUCTION

Deep learning models have been applied in many real world applications such as classification [4,5], object detection [6,7], and time series forecasting [8,9]. Their performances on these applications have been outstanding. In addition to these applications, the models can learn from different data types such as graph structured data. However, these models are challenged when they are expected to perform previously learned tasks while learning a current one. This is called continual learning (CL) and has become a hot topic with the recent developments in deep learning.

CL research focuses on training a model such that it can perform multiple tasks without forgetting the previously learned ones. However, the main challenge in CL is the catastrophic forgetting [10] which proposes that the trained models tend to lose previously learned information to learn the current task. This requires training different models for different tasks which is unfeasible in terms of both computation wise and memory wise. The CL research focuses on overcoming this problem and in this thesis, we addressed continual learning on graph structured data.

In the modern world, a large amount of data is generated every day, and graph structured data is not an exception. Social networks can be an example for the changing graph structured data where the interactions are continuous and new users are emerging every day. Drug design is an example where new molecules and properties of molecules may encountered [11] and the drug design process must be updated accordingly. Therefore, continual learning on graph structured data, or graph continual learning (GCL), is required so that the models can be adapted to expanding graphs or emerging new categories within the graphs [12].

GCL research has gained significant interest with the increase in data amount. Although the benchmarks from CL research have been inherited [13]–[15], these benchmarks must be adapted to GCL by taking the graph specific properties. In addition to the previously constructed CL benchmarks, some studies [11,16] have

addressed GCL. There is much to discover in the GCL field as it progresses with the CL field.

As mentioned earlier, the CL field focuses on learning incoming tasks without forgetting the previous ones. Many approaches such as replay buffer [17] and regularization [18] are employed in CL, and meta learning is an approach that can also benefit the CL. Since meta learning aims at a model to generalize the incoming tasks [3], meta learning guides the model parameters so that the model parameters would not fit the optimal parameters for any specific task. Therefore, meta learning can be applied in the CL field. Besides its usage in the CL field, meta learning can also be applied to different data types such as text data [19,20] and graph data [21,22].

In this thesis, we propose a GCL approach, MetaCLGraph, which merges meta learning and replay buffer with learnable learning rates on continual learning on graph structured data. Our experiments indicate that the proposed MetaCLGraph method outperforms both the classical CL methods and the GCL methods. We also investigated our model by investigating the effects of replay buffer and meta learning.

2. BACKGROUND AND LITERATURE REVIEW

2.1 Graph Neural Networks

Working with graph structured data is different than working with visual data. First of all, graph structured data consists of vertices (or nodes) and edges. An example for graph is provided in Figure 2.1. The edges connect the vertices while nodes represent the items. Each node can obtain attributes associated with it so that it provides more information about the item. In addition, graph-structure data is non-Euclidean so it does not have any specific hierarchy and order [23]. Therefore, graph-structure data does not have any certain geometry and hence, this difference requires a different type of neural network so that the graph structured data can be studied properly.

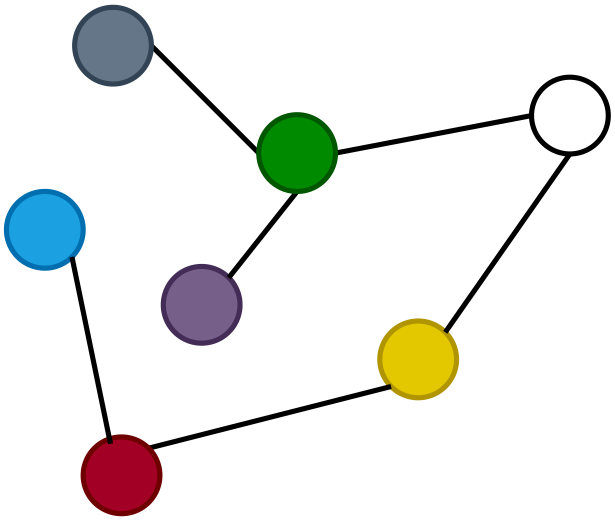

Figure 2.1 : An example graph - Self drawn.

GNNs are constructed to process the graph-structured data and their inference problems [24]. GNNs use the graph structure and node features to learn representations of nodes or the entire graph [2]. The relations inside the graph and the properties of the graph members are discovered with the utilization of GNN. Therefore, GNNs can perform different tasks on graphs such as node classification and link prediction.

To achieve graph related tasks like node classification and link prediction, GNNs utilize embeddings and message passing so that both the graph structure and the node properties can be learned. Embeddings and message passing are utilized so that GNNs can perform on the graph structured data. Embeddings and message passing are discovered in the following paragraphs.

2.1.1 Embeddings

Embeddings are utilized to represent the graph structured data in an embedding space. They are the low-dimensional vector representations of nodes or graphs. Representation of a node embedding is shown in Figure 2.2. To obtain embeddings, an encoder is employed as a mapping function and each node is mapped into a low-dimensional vector [1]. This process aims both to decrease the dimensionality of the data and obtain accurate representations of the nodes so that GNNs can discover the graph data properly.

2.1.2 Message passing

As mentioned earlier, the graph-structured data consists of nodes and edges and the edges are the connections between the nodes. Since the connections between the nodes may indicate a similarity between connected nodes, these connections and the connected nodes must be evaluated. While discovering the effects of neighboring nodes, the information withheld in the node itself must be preserved and evaluated accordingly. Therefore, message passing becomes important for GNNs and it also establishes the basis concept for the GNN layers. Message passing allows GNNs to consider the underlying similarities between the nodes and discover more information by considering neighboring information.

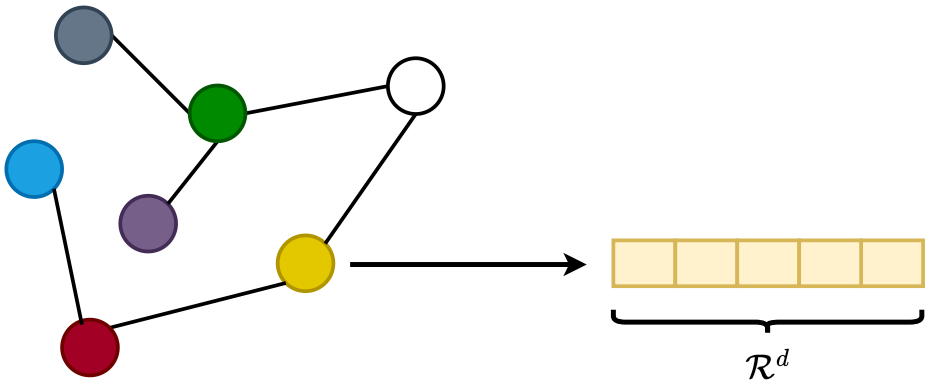

Figure 2.2 : The representation of a node as node embedding [1] - Self drawn.

Message passing in GNNs takes place in two steps. First, each node in the graph constructs a message using an employed message function. The message function constructs the message which will be sent to other nodes for their calculation. After constructing messages for all nodes, these messages are aggregated using an aggregation function [1]. Although there may be several aggregation functions used in GNNs, it is important to maintain the properties of the node itself since the properties of neighboring nodes are only used for understanding the relationship withheld.

The inner mechanism of a GNN layer is shown in Figure 2.3. Given a graph $\mathscr{G} = (\mathscr{V}, \mathscr{E})$, where $\mathscr{V} = \{v_i\}_{i=1}^{|\mathscr{V}|}$ denotes the set of nodes and $\mathscr{E} = \{e_{ij}\}_{i,j=1}^{|\mathscr{E}|}$ denotes the edges of the graph, GNNs operate in two phases to compute the output embeddings $\mathbf{h}_v^{(l)}$. First, the embeddings are obtained by multiplying the weight matrices of each layer $\mathscr{W}^{(l)}$ with the embeddings 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 GNNs to preserve the information from node v.

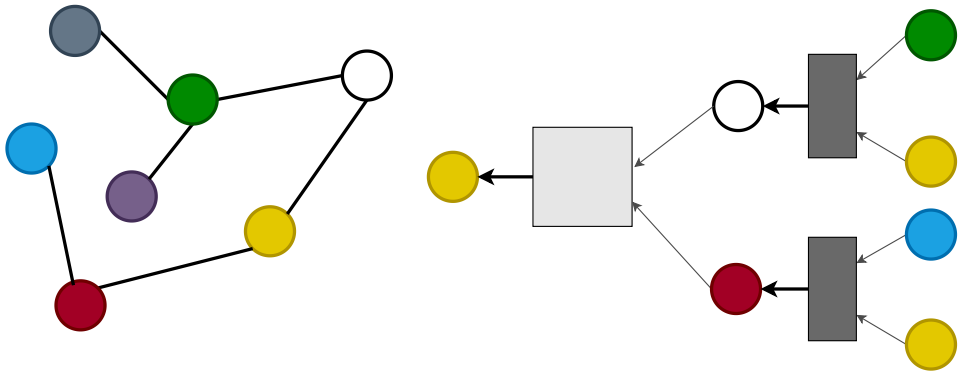

Figure 2.3 : The representation of a GNN layer showing message passing [2] - Self drawn.

2.2 Meta Learning

Meta learning refers to processes in which a model learns how to learn. The aim is to provide a model with the capability to generalize itself to new tasks. The meta learning paradigm guides the model weights so that the model can find the set of parameters that can collectively adapt to all observed tasks. MAML [3] is one of the most popular meta learning algorithms due to its model-agnostic nature. The diagram for MAML is given in Figure 2.4.

The MAML algorithm operates as follows. 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 D_i^{tr} and D_i^{ts} represent the training and test set for task i, respectively.

$$\min_{\theta} \sum_{\text{task } i} L\left(\theta - \alpha \nabla_{\phi} L\left(\theta, \mathscr{D}_i^{\text{tr}}\right), \mathscr{D}_i^{\text{ts}}\right). \tag{2.1}$$

Meta learning ensures that the model does not converge to the optimal parameters of a single task and preserves the adaptability of the model to incoming tasks. This allows meta learning to be used in different fields such as reinforcement learning [3,25], few-shot learning [26], and continual learning [27].

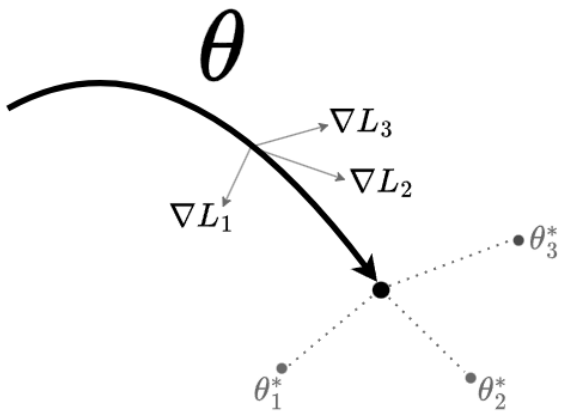

Figure 2.4 : The diagram for model-agnostic meta learning [3] - Self drawn.

2.3 Continual Learning

CL is a machine learning approach where the challenge is that a constructed learning model executes incoming tasks while maintaining its performance over the earlier tasks. The deep learning models tend to provide poor results when they are expected to learn from a sequence of data or tasks [28]. In CL, 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 CL is called catastrophic forgetting [10]. Catastrophic forgetting causes models to lose the previously obtained information about observed tasks and not perform those tasks due to the newly obtained information regarding the current task. Since the aim of CL is to obtain a model that can also perform previously observed tasks, this issue must be resolved. To tackle this issue, two CL setups are generally used, task incremental and class incremental [29], respectively. Although both setups share the same goal of adapting incoming tasks without losing previous information, the model has a task indicator for task incremental setup, different

from class incremental setup [12]. Therefore, the model does not consider every observed task so far, instead, it performs according to the identified task by the task indicator. However, class incremental setup requires the model to distinguish among all classes from every observed task so far. Therefore, the class incremental setup becomes more challenging compared to the task incremental setup. There are three main approaches to dealing with catastrophic forgetting, namely parameter isolation methods, regularization based methods, and replay based methods.

Parameter isolation methods dedicate parameters for certain tasks so that when a new task arrives, previously dedicated tasks are isolated from the learning process. This allows the model to preserve the information from the previous tasks by assigning some of its parameters to those tasks specifically. Meanwhile, unassigned parameters can be used to learn the incoming tasks. The parameters are assigned accordingly to their update during the training stage. For instance, [30] employs a task based hard attention mechanism which learns a mask after each task. This mask allows model to preserve weights so that the performance can be maintained.

Regularization based methods employ a regularization term inside the loss function so that the model can achieve CL. Introducing a regularization term inside the loss function can be useful to mitigate the forgetting of previous tasks since the regularization term can be employed to maintain previously obtained information. Regularization based methods include [18] and [17] where [18] employs a regularization term obtained from the Fisher information matrix to approximate the importance of each model parameter while [17] uses synapses to determine the importance of model parameters for each task.

Replay based methods employ a replay structure to store examples from previous tasks. The stored examples are used during the training of the incoming tasks. Since the model tends to adapt to the current task, introducing examples from earlier tasks during the training of the model is helpful for the model parameters so that the update on the model would not be solely based on the current task and the information from previous tasks can be maintained. There are two types of replays, namely rehearsal replays and pseudo-rehearsal replays. Rehearsal replays store examples from previous

tasks to use in future tasks. Although rehearsal replays allow the models to observe previous examples, storing examples raises issues in terms of data privacy. To address the privacy issues, pseudo-rehearsal replays are used where example samples are generated by using a generative model. [31] is an example for replay based methods where a memory buffer is employed to store examples from previous tasks while [32] is an example for pseudo-rehearsal replay where a generative adversarial network (GAN) is used to generate items from earlier tasks.

Although the setups and methods for CL were initially developed on image data, they can be adapted to graph-structured data as graph continual learning (GCL) started to emerge as a research field. With the increasing real world applications including citation networks and social networks, GCL has become an important research area. The appearance of new items and categories requires a model to be updated so that new items can be learned [11]. In addition, many graph data is generated continuously [12] so it is necessary to adapt continual learning methods to graph machine learning.

The evaluation metrics of GCL methods are similar to the CL methods. The most common metrics are namely average performance (AP) and average forgetting (AF), [31]. While AP evaluates the overall performance of the model on all tasks, AF focuses on the model's ability to deal with catastrophic forgetting. The formulation of AP and AF are given as follows where $P_{i,j}$ represents the accuracy at the ith task after observing j tasks.

$$AP = \frac{\sum_{i=1}^{T} P_{i,T}}{T}, AF = \frac{\sum_{i=1}^{T-1} P_{i,T} - P_{i,i}}{T-1} \tag{2.2}$$

While adapting continual learning into graph machine learning, the properties of graph structured data must be taken into account. For instance, the adjacency matrix needs to be taken into account since it shows the connections among the graph data. Therefore, the earlier CL methods are adapted while considering the properties of the graph data. In addition to applying earlier methods, specific GCL methods can also be constructed by using graph-specific properties. For instance, GNNs benefit from the topological structure of graphs [11], therefore it can be used in GCL research.

3. META CONTINUAL LEARNING ON GRAPHS WITH EXPERIENCE REPLAY

3.1 Introduction to Meta Continual Learning on Graphs with Experience Replay

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 [28]. 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 [10,33], which causes the model to lose obtained information from earlier tasks, leading to a 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 [14,31] or generate pseudo samples using the stored information [32,34]. The second group focuses on parameter isolation from earlier tasks such that the model can preserve its performance on earlier tasks [30,35], 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 [18,28]. 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 [29]. In class incremental learning, the individual classes are presented sequentially [36] while only the tasks are presented in task incremental learning [29]. 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 [12]. In addition, class incremental 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 [3] 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 [37,38]. 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 [21,22].

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 [23]. 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 and, 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.

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 $\mathscr{B}$
    Calculate loss with $T_{t'}$
  end for
  Sum Losses from T_t & $\mathscr{B}$
  Extend $\mathscr{B}$ 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**
  $\mathscr{G}_{aux} \leftarrow \mathscr{B} \cup T_t$
  Calculate weights with T_t
  Meta loss with calculated weights on $\mathscr{G}_{aux}$
  Update learning rates
  Extend $\mathscr{B}$ with T_t samples
end for

Continual learning on graphs has been studied recently [11,12,16] and it has become an emerging field. While the studies addressing this field has similar approaches such as memory-based [16] or regularization-based [11], they also consider the properties of the graph structured data as their methods can be affected by different factors of the graph structure [39] such as topology and irregularity of the graph. In addition, newly arriving tasks alter the dynamics of the graph, interfering with the learning of the model. Therefore, continual learning on graphs diverges and it is handled concerning both the continual learning paradigm and the characteristics of the graph structured data.

In our research, we fused 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 Algorithms 1, 2, and 3, respectively. Our developed method fuses the experience replay & selection method in the ER-GNN [16] with the meta learning & learnable learning rates from the LA-MAML [27] to advance on graph continual learning. 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.

3.2 Related Work

Continual learning mainly deals with the catastrophic forgetting problem. In order to overcome the latter, three approaches are generally used aiming to maintain the adaptability of the model to incoming tasks without any performance loss. 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 [30,35]. 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 [17,18]. 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 [16,31]. Capacity saturation is also another challenge in continual learning [40]. The architecture of the model defines the capacity as the components of the architecture affect learning dramatically [41,42]. A model with fixed capacity saturates as it is kept training on more tasks since the model loses its ability to adapt to incoming tasks. This is called the stability-plasticity dilemma [43]. The same dilemma exists in graph continual learning where for GNN architectures, the learning may become limited as a large number of new classes arrives. Addressing this dilemma requires a specialized focus, and while numerous studies addressed the effects of architecture on continual learning [44,45], a similar study on GNNs has yet to be conducted.

Graph Neural Networks recently have been used in continual learning on graphs [46]. Since GNNs consider both the adjacency matrix and the features as mentioned earlier, they are capable of discovering the relationship between the nodes and the knowledge held by the graph. Graph continual learning is a very recent topic [16] where GNNs are employed for continual learning of graph-structured data. General continual learning approaches, namely regularization based and replay based methods, are adapted to graph continual learning. Inspired by the work in the continual learning field, replay based methods are developed for graph continual learning, storing samples from earlier tasks [16] or generating samples by using the stored representations from earlier tasks [47]. On the other hand, regularization based methods use an additional term in the loss function to adjust the updates in the model weights. In graph continual learning, regularization based methods [11,48]–[50] use a regularization term to handle the issues caused by the properties of the graph structured data such as topology and irregularity. 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 [51]. Due to the nature of the real-world scenarios, dynamic graph learning [52,53] can be applied to capture the dynamics of the real-world graphs. Whereas dynamic graph learning primarily concentrates on capturing the up-to-date graph representations [54] and it does not focus on the forgetting problem because the observed data can still be accessed by the model. However, in graph continual learning, the observed data is not accessible once after the model is trained on that task, requiring the model to develop mechanisms to deal with forgetting and use the obtained knowledge. Due to the nature of the real-world graphs, the GNN model becomes more powerful with continual learning, as the same GNN model can be used for different tasks. This allows us to avoid training separate models for each task, and obtain a single model that can perform all tasks which is in line with the ultimate goal of lifelong learning.

The implementation of memory mechanisms proves to be really effective [55] as memory mechanisms allow models to revisit the earlier examples during the current task and maintain the performances from earlier tasks. Various types of memory units are explored for 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 [16,31]. 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 [14]. 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 in reinforcement learning and few-show learning [3] to perform multi task learning on a large set of tasks. Since the aim of continual learning is to adjust the model to newly coming tasks while maintaining its ability to perform in older ones, meta learning becomes useful in continual learning. Meta learning prevents overfitting for a specific task by using the gradients of optimal parameters for each task [27]. This approach is effective in mitigating catastrophic learning as it prevents the model from overfitting to a specific task and the learning process of the model would not be stopped.

Our proposed approach, namely Meta Continual Learning for Graphs with Experience Replay (MetaCLGraph), fuses meta learning with a memory mechanism to increase the efficiency in learning incoming tasks. Focused on the node classification problem in the class incremental learning setting, MetaCLGraph uses a meta learning mechanism to improve the model's ability to store already-seen information while learning the current task and incorporates a replay buffer as the memory mechanism that 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.

3.3 Method

3.3.1 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 [36].

3.3.2 MetaCLGraph

MetaCLGraph is our proposed solution for graph continual learning which fuses meta learning and episodic memory through an experience replay mechanism. The algorithm for our method is given in Algorithm 4. Although [16] and [27] both inspire the MetaCLGraph, the transition to the latter is not straightforward, as the adaptation of the LA-MAML to graph continual learning while using an episodic memory includes overcoming the following obstacles such as lack of graph nodes batches and utilizing the replay buffer efficiently. The model weights are initially calculated for the current task and later updated using gradients calculated while using both stored examples and current task data. This update is called the meta update and it allows instead of fitting for just the current task T_i, to be able to learn the previous tasks along with current task T_i. The experience replay is obtained when the model is trained on a task and some nodes are selected by using the coverage maximization function (3.3.3). 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.

Algorithm 4 MetaCLGraph-Detailed Algorithm

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

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

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.

3.3.3 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 [16] to determine the nodes which can represent their tasks accordingly. Coverage maximization method is used as the selection function to maximize the coverage of the embedding space by finding the distances of nodes to the other class. According to the distances, the nodes are ranked and lower ranked nodes are selected. The equation for the coverage maximization is given in the following.

$$\mathscr{N}(v_i) = \left\{ v_j \mid \text{dist}\left(v_i - v_j\right) < d, \mathscr{Y}(v_i) \neq \mathscr{Y}\left(v_j\right) \right\} \tag{3.1}$$

where $\mathcal{N}(v_i)$ is the set of nodes coming from different classes with distance d to node v_i, $\mathcal{Y}(v_i)$ and $\mathcal{Y}(v_j)$ are the labels of the nodes v_i and v_j, respectively. The distance inside the embedding space is minimized so that the classes can be represented inside the buffer. This reinforces the model in remembering the previous classes as the selected examples in this manner cover their embedding spaces.

The experience replay allows us to construct a merged dataset as illustrated in Figure 3.1. 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.

3.3.4 Meta learning

The meta learning is adapted to MetaCLGraph in two steps as follows. When learning a task T_i, the model weights for the current task T_i, which are called fast weights θ_{fast} are calculated using the current task data, shown in Algorithm 4, line 4. This step is called the inner update and allows the model to benefit from the direction of the gradients for the current task. After the inner update, θ_{fast} is introduced to the model, and the meta loss is calculated when the model is fed with the merged dataset which contains the current task data and the buffer samples from previous tasks. This is called the meta update and the model weights θ are updated with the gradients with respect to the meta loss using θ_{fast} in this step, shown in Algorithm 4, line 6. At the end of the meta learning, the model is updated considering the gradients with respect to the current task data and also the samples from previous tasks.

3.3.5 Learning rate updates

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.

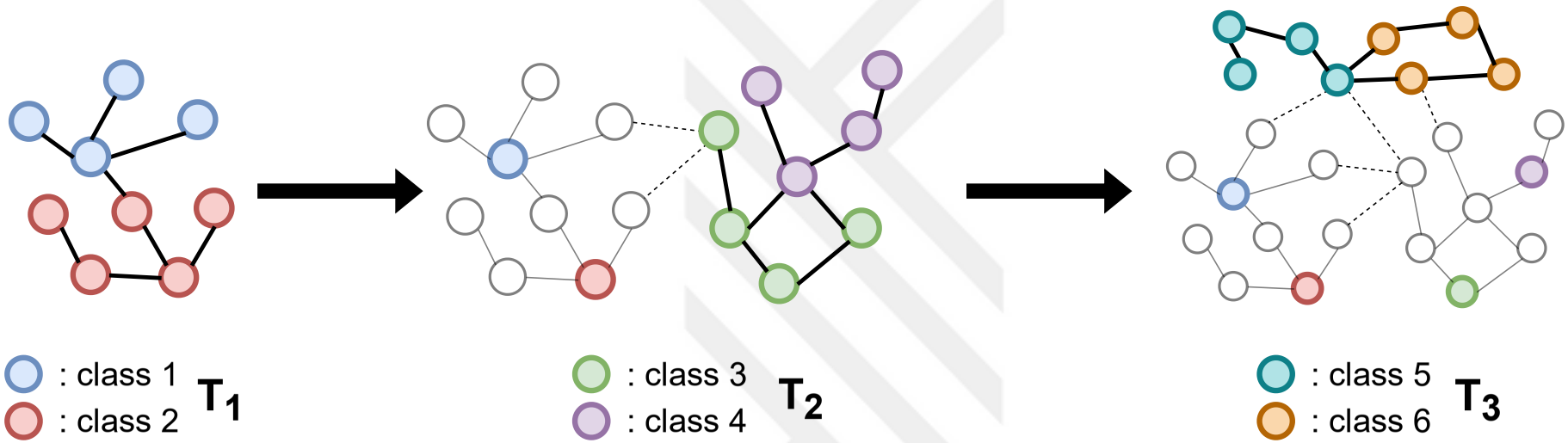

Figure 3.1 : The dataset merging process is illustrated in this figure. The arrows represent the transition between the tasks while coloured nodes represent the active nodes during that task. In our approach, multiple nodes are selected at the end of the task and stored inside the replay buffer. When a new task is arrived, stored nodes are merged with the current task and form a merged dataset.

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 [27]. 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. The computation of the gradients with respect to learning rates are given in Appendix A [27].

3.4 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 [12] is used for the experiments. The datasets, baselines, experimental settings, and results are reported and described next. We provide a reference implementation of the proposed model and the experimental pipeline.

3.4.1 Datasets

For evaluating MetaCLGraph, four benchmark datasets were employed: Corafull [56], Arxiv [57], Reddit [58], and Citeseer [59]. Arxiv, Corafull, and Reddit are used in [12] to construct a benchmark for continual graph learning, and Citeseer is additionally included for its significance in graph structured data. In this problem, we divide the datasets into tasks containing two classes for each task. This resulted in 35 tasks from Corafull, 3 from Citeseer, and 20 each from Arxiv and Reddit. For the Reddit dataset, the 41^{st} and the last class is dropped since it only has one example. Table 3.1 provides the detailed task information, and graph structures for each dataset.

Table 3.1 : 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

To manage device memory requirements, larger graphs like Reddit are divided into batches, while the rest are processed as single batches.

3.4.2 Baselines

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

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) [18] focuses on preserving crucial model weights for each task, allowing updates on only less significant weights during the learning of the new tasks to preserve earlier information.

Memory aware synapses (MAS) [17] 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) [11] is a method introduced for graph continual learning which integrates weight preservation considering the graph topology. 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 [16] 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 combines meta learning with experience replay.

Joint is the method where all the learned tasks are accessed during training. Therefore, it serves as 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.

3.4.3 Experimental 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 [60] is selected as the optimizer. All methods use the graph convolutional network as the backbone GNN architecture [24]. The selection algorithm relies on coverage maximization. The hyperparameters concerning the compared methods are obtained from the benchmark paper [12] 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 [12], 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) [31]. AP focuses on the model's accuracy in classification of each task. It is obtained 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:

$$a_{j,j} - a_{k,j}, \quad \forall j < k \tag{3.2}$$

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.

3.4.4 Results

The experimental results are reported in Table 3.2. 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. On the other hand, the MetaCLGraph has the best AP score among the baselines while its AF score is the second best. This shows that the MetaCLGraph can learn the tasks for the Reddit dataset while preserving its performance for the previous tasks.

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 and the model can generalize the observed tasks with the meta loss. In addition to meta learning, the MetaCLGraph method also utilizes the learnable learning rates which increases the model's ability to learn incoming tasks, compared to ER-GNN. For the Corafull dataset, ER-GNN forgets the previous tasks to learn the current task properly, causing its AP score to decrease dramatically. Meanwhile, the MetaCLGraph method preserves the obtained information by using learnable learning rates. Hence, the MetaCLGraph method outperforms ER-GNN with the utilization of meta learning and learnable learning rates.

In Figures 3.2 and 3.3, 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.

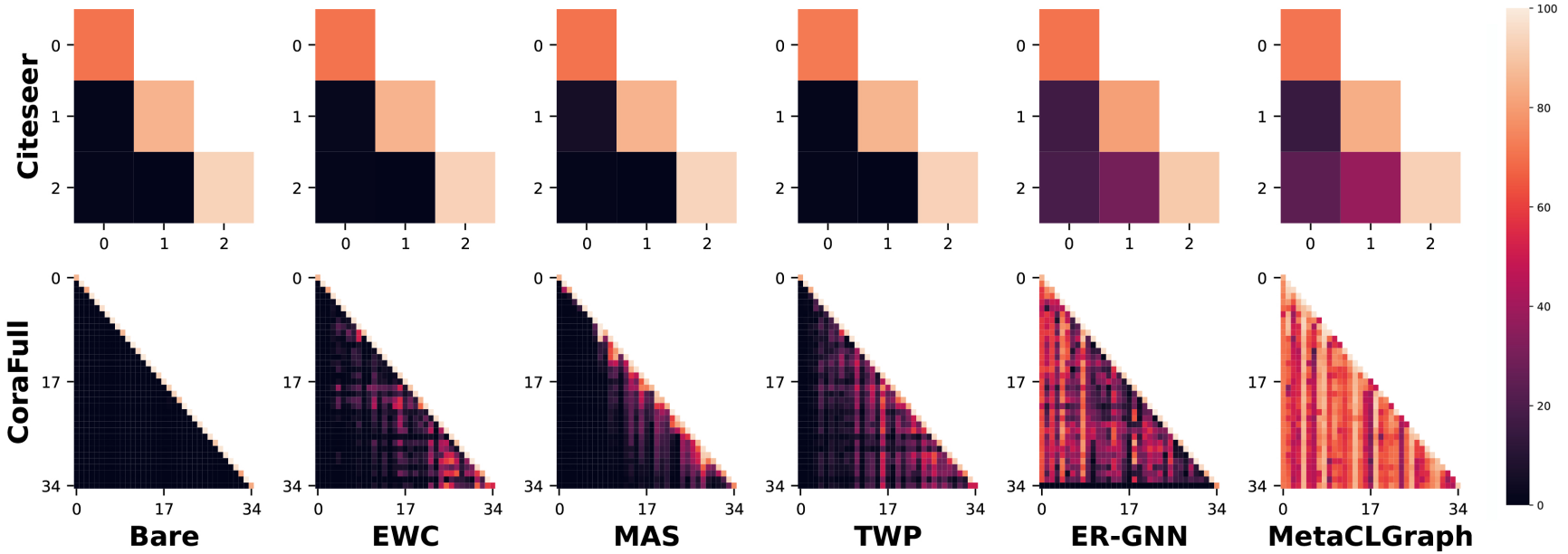

Figure 3.2 : The visualization of performance matrices for MetaCLGraph and benchmark methods for the Citeseer and Corafull datasets. Each entry in the matrices represents the performance of the method on the column task while learning the task on the row. The determined performance metric on the visualization is accuracy where lighter colours represent the higher score while darker colours represent the lower scores.

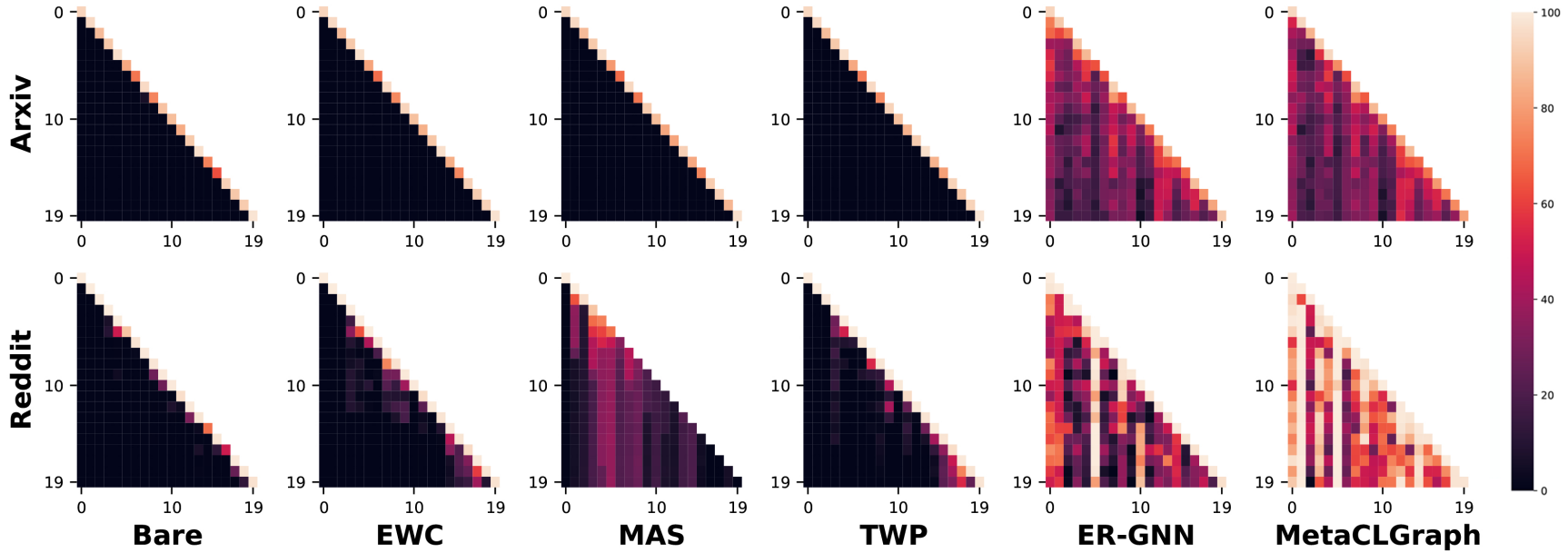

Figure 3.3 : The visualization of performance matrices for MetaCLGraph and benchmark methods for the Arxiv and Reddit datasets. Each entry in the matrices represents the performance of the method on the column task while learning the task on the row. The determined performance metric on the visualization is accuracy where lighter colours represent the higher score while darker colours represent the lower scores.

Table 3.2 : 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	-

3.4.5 Computational costs

The computational costs are also investigated and the costs are reported in Table 3.3. The experiments are repeated 5 times on the Corafull dataset. The results show that the computational costs of our proposed model surpass the baseline models. Although the computational costs of our model are the highest among the baselines, its computational costs are justifiable since its performance is beyond other methods and the computational costs are tested on the dataset with the greatest number of tasks among the tested datasets. Corafull dataset contains the greatest number of tasks among the benchmark datasets, showing that the application of our model to other datasets would not be limited since the computational cost difference among the baselines would not be greater.

Table 3.3 : The computational cost table for various baselines and MetaCLGraph.

Method	*Time per experiment (s)*
Base method	159.77
EWC	381.46
TWP	418.71
MAS	177.50
ER-GNN	234.87
MetaCLGraph	616.28

3.4.6 Ablation study

This section focuses on the impact of meta learning and experience replay on the performance of MetaCLGraph in class incremental learning. MetaCLGraph variants that either exclude meta learning or experience replay are examined to understand their contributions to the model performance. 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 3.4.4.

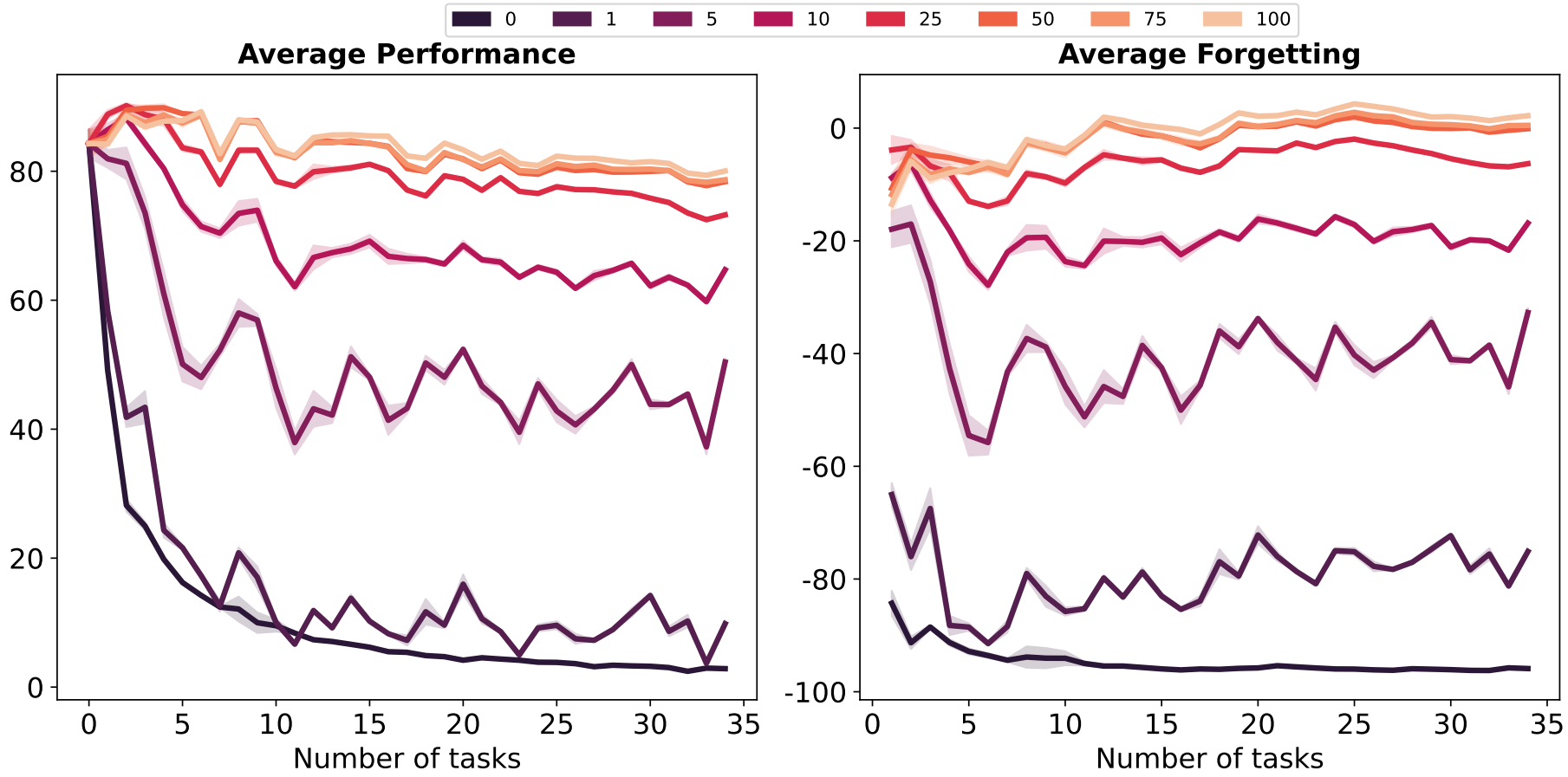

Figure 3.4 : The effect of buffer budget on MetaCLGraph is shown for AP (left) and AF (right). As the buffer budget increases, both the model performance and forgetfulness are improved due to the increase in the stored number of samples per class.

The results for the ablation study are given in Figure 3.4. 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 trained more than specified epochs and the model's performance on that task is improved by meta learning and additional epochs. Therefore, a limit on the number of stored examples in the buffer is imposed so that the integrity of continual learning can be maintained.

3.5 Broad Impact, Limitations and Future Work

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. Another limitation is the increase in the number of tasks as it may limit the application of the proposed model since the overall complexity of the model is already higher than its counterparts.

4. CONCLUSION

In this thesis, we proposed a novel method for continual learning on graph structured data. We built a novel method that merges replay buffer and meta learning with learnable learning rates and performs continual learning on graph structured data.

Our proposed method utilizes the generalization of tasks from meta learning while the previously stored examples inside the buffer are used to maintain the previously learned information by the model. In addition, the learnable learning rates are employed to improve the learning of the model by adjusting the updates on the model weights. In the experiments, it is shown that our method outperforms both CL methods adapted to the graph domain and the GCL methods in class incremental CL setup at node classification task. In addition, we showed the effects of replay buffer and meta learning separately on continual learning with ablation studies. In addition, we also investigated the effect of the replay buffer capacity with different capacities.

Some future directions that can be led by the thesis are given as follows.

- We employed MetaCLGraph for the node classification task in class incremental learning setup. The setup and the task for MetaCLGraph can be altered for future work.
- Our proposed method showed that the utilization of meta learning allows replay buffer to become more effective. Changing the memory mechanism could be another future direction since storing examples from earlier tasks may cause problems in terms of data privacy.
- The capacity of the replay buffer is also discussed in this study. Another future direction could be the limitations caused by the memory capacity and the architecture so that capacity saturation can be covered thoroughly.

- We determined that increasing the replay buffer capacity may contradict the purpose of CL as the buffer may store the entire task if the data size for a task is small. A future direction could be investigating the effect of memory mechanisms on learning when they are storing all of the previous task data.

REFERENCES

[1] **Hamilton, W.L.**, **Ying, R. and Leskovec, J.** (2017). Representation learning on graphs: Methods and applications, *arXiv preprint arXiv:1709.05584*.

[2] **Xu, K.**, **Hu, W.**, **Leskovec, J. and Jegelka, S.** (2018). How powerful are graph neural networks?, *arXiv preprint arXiv:1810.00826*.

[3] **Finn, C.**, **Abbeel, P. and Levine, S.** (2017). Model-agnostic meta-learning for fast adaptation of deep networks, *International conference on machine learning*, PMLR, pp.1126–1135.

[4] **Hochreiter, S. and Schmidhuber, J.** (1997). Long short-term memory, *Neural computation*, *9*(8), 1735–1780.

[5] **Zhang, K.**, **Huang, Y.**, **Du, Y. and Wang, L.** (2017). Facial expression recognition based on deep evolutional spatial-temporal networks, *IEEE Transactions on Image Processing*, *26*(9), 4193–4203.

[6] **Ren, S.**, **He, K.**, **Girshick, R. and Sun, J.** (2015). Faster r-cnn: Towards real-time object detection with region proposal networks, *Advances in neural information processing systems*, *28*.

[7] **Redmon, J.**, **Divvala, S.**, **Girshick, R. and Farhadi, A.** (2016). You only look once: Unified, real-time object detection, *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp.779–788.

[8] **Lai, G.**, **Chang, W.C.**, **Yang, Y. and Liu, H.** (2018). Modeling long-and short-term temporal patterns with deep neural networks, *The 41st international ACM SIGIR conference on research & development in information retrieval*, pp.95–104.

[9] **Bai, S.**, **Kolter, J.Z. and Koltun, V.** (2018). An empirical evaluation of generic convolutional and recurrent networks for sequence modeling, *arXiv preprint arXiv:1803.01271*.

[10] **McCloskey, M. and Cohen, N.J.**, (1989). Catastrophic interference in connectionist networks: The sequential learning problem, Psychology of learning and motivation, volume 24, Elsevier, pp.109–165.

[11] **Liu, H.**, **Yang, Y. and Wang, X.** (2021). Overcoming catastrophic forgetting in graph neural networks, *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pp.8653–8661.

[12] **Zhang, X.**, **Song, D. and Tao, D.** (2022). CGLB: Benchmark Tasks for Continual Graph Learning, *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

[13] **Lin, Z.**, **Shi, J.**, **Pathak, D. and Ramanan, D.** (2021). The clear benchmark: Continual learning on real-world imagery, *Thirty-fifth conference on neural information processing systems datasets and benchmarks track (round 2)*.

[14] **Rebuffi, S.A.**, **Kolesnikov, A.**, **Sperl, G. and Lampert, C.H.** (2017). icarl: Incremental classifier and representation learning, *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp.2001–2010.

[15] **Prabhu, A.**, **Torr, P.H. and Dokania, P.K.** (2020). Gdumb: A simple approach that questions our progress in continual learning, *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*, Springer, pp.524–540.

[16] **Zhou, F. and Cao, C.** (2021). Overcoming catastrophic forgetting in graph neural networks with experience replay, *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp.4714–4722.

[17] **Aljundi, R.**, **Babiloni, F.**, **Elhoseiny, M.**, **Rohrbach, M. and Tuytelaars, T.** (2018). Memory aware synapses: Learning what (not) to forget, *Proceedings of the European conference on computer vision (ECCV)*, pp.139–154.

[18] **Kirkpatrick, J.**, **Pascanu, R.**, **Rabinowitz, N.**, **Veness, J.**, **Desjardins, G.**, **Rusu, A.A.**, **Milan, K.**, **Quan, J.**, **Ramalho, T.**, **Grabska-Barwinska, A.** *et al.* (2017). Overcoming catastrophic forgetting in neural networks, *Proceedings of the national academy of sciences*, *114*(13), 3521–3526.

[19] **Dou, Z.Y.**, **Yu, K. and Anastasopoulos, A.** (2019). Investigating meta-learning algorithms for low-resource natural language understanding tasks, *arXiv preprint arXiv:1908.10423*.

[20] **Bansal, T.**, **Jha, R.**, **Munkhdalai, T. and McCallum, A.** (2020). Self-Supervised Meta-Learning for Few-Shot Natural Language Classification Tasks, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp.522–534.

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

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

[23] **Asif, N.A.**, **Sarker, Y.**, **Chakrabortty, R.K.**, **Ryan, M.J.**, **Ahamed, M.H.**, **Saha, D.K.**, **Badal, F.R.**, **Das, S.K.**, **Ali, M.F.**, **Moyeen, S.I.**, **Islam, M.R. and Tasneem, Z.** (2021). Graph Neural Network: A Comprehensive Review on Non-Euclidean Space, *IEEE Access*, *9*, 60588–60606.

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

[25] **Rakelly, K.**, **Zhou, A.**, **Finn, C.**, **Levine, S. and Quillen, D.** (2019). Efficient off-policy meta-reinforcement learning via probabilistic context variables, *International conference on machine learning*, PMLR, pp.5331–5340.

[26] **Snell, J.**, **Swersky, K. and Zemel, R.** (2017). Prototypical networks for few-shot learning, *Advances in neural information processing systems*, *30*.

[27] **Gupta, G.**, **Yadav, K. and Paull, L.** (2020). Look-ahead meta learning for continual learning, *Advances in Neural Information Processing Systems*, *33*, 11588–11598.

[28] **Li, Z. and Hoiem, D.** (2017). Learning without forgetting, *IEEE transactions on pattern analysis and machine intelligence*, *40*(12), 2935–2947.

[29] **De Lange, M.**, **Aljundi, R.**, **Masana, M.**, **Parisot, S.**, **Jia, X.**, **Leonardis, A.**, **Slabaugh, G. and Tuytelaars, T.** (2021). A continual learning survey: Defying forgetting in classification tasks, *IEEE transactions on pattern analysis and machine intelligence*, *44*(7), 3366–3385.

[30] **Serra, J.**, **Suris, D.**, **Miron, M. and Karatzoglou, A.** (2018). Overcoming catastrophic forgetting with hard attention to the task, *International conference on machine learning*, PMLR, pp.4548–4557.

[31] **Lopez-Paz, D. and Ranzato, M.** (2017). Gradient episodic memory for continual learning, *Advances in neural information processing systems*, *30*.

[32] **Atkinson, C.**, **McCane, B.**, **Szymanski, L. and Robins, A.** (2018). Pseudo-recursal: Solving the catastrophic forgetting problem in deep neural networks, *arXiv preprint arXiv:1802.03875*.

[33] **Goodfellow, I.J.**, **Mirza, M.**, **Xiao, D.**, **Courville, A. and Bengio, Y.** (2013). An empirical investigation of catastrophic forgetting in gradient-based neural networks, *arXiv preprint arXiv:1312.6211*.

[34] **Lavda, F.**, **Ramapuram, J.**, **Gregorova, M. and Kalousis, A.** (2018). Continual classification learning using generative models, *arXiv preprint arXiv:1810.10612*.

[35] **Mallya, A.**, **Davis, D. and Lazebnik, S.** (2018). Piggyback: Adapting a single network to multiple tasks by learning to mask weights, *Proceedings of the European conference on computer vision (ECCV)*, pp.67–82.

[36] **Masana, M.**, **Liu, X.**, **Twardowski, B.**, **Menta, M.**, **Bagdanov, A.D. and Van De Weijer, J.** (2022). Class-incremental learning: survey and performance evaluation on image classification, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *45*(5), 5513–5533.

[37] **Nguyen, B.D.**, **Do, T.T.**, **Nguyen, B.X.**, **Do, T.**, **Tjiputra, E. and Tran, Q.D.** (2019). Overcoming data limitation in medical visual question answering, *Medical Image Computing and Computer Assisted Intervention–MICCAI 2019: 22nd International Conference, Shenzhen, China, October 13–17, 2019, Proceedings, Part IV 22*, Springer, pp.522–530.

[38] **Chen, L.**, **Jose, S.T.**, **Nikoloska, I.**, **Park, S.**, **Chen, T.**, **Simeone, O.** *et al.* (2023). Learning with limited samples: Meta-learning and applications to communication systems, *Foundations and Trends® in Signal Processing*, *17*(2), 79–208.

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

[40] **Sodhani, S.**, **Chandar, S. and Bengio, Y.** (2020). Toward training recurrent neural networks for lifelong learning, *Neural computation*, *32*(1), 1–35.

[41] **Mirzadeh, S.I.**, **Chaudhry, A.**, **Yin, D.**, **Nguyen, T.**, **Pascanu, R.**, **Gorur, D. and Farajtabar, M.** (2022). Architecture matters in continual learning, *arXiv preprint arXiv:2202.00275*.

[42] **Shahawy, M.**, **Benkhelifa, E. and White, D.** (2022). A Review on Plastic Artificial Neural Networks: Exploring the Intersection between Neural Architecture Search and Continual Learning, *arXiv preprint arXiv:2206.05625*.

[43] **Mermillod, M.**, **Bugaiska, A. and Bonin, P.** (2013). *The stability-plasticity dilemma: Investigating the continuum from catastrophic forgetting to age-limited learning effects*.

[44] **Huang, S.**, **Francois-Lavet, V. and Rabusseau, G.** (2021). Understanding Capacity Saturation in Incremental Learning., *Canadian Conference on AI*.

[45] **Feillet, E.**, **Petit, G.**, **Popescu, A.**, **Reyboz, M. and Hudelot, C.** (2023). AdvisIL-A Class-Incremental Learning Advisor, *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp.2400–2409.

[46] **Wang, J.**, **Song, G.**, **Wu, Y. and Wang, L.** (2020). Streaming graph neural networks via continual learning, *Proceedings of the 29th ACM international conference on information & knowledge management*, pp.1515–1524.

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

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

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

[50] **Sun, L.**, **Ye, J.**, **Peng, H.**, **Wang, F. and Philip, S.Y.** (2023). Self-supervised continual graph learning in adaptive riemannian spaces, *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp.4633–4642.

[51] **Tang, B. and Matteson, D.S.** (2020). Graph-based continual learning, *arXiv preprint arXiv:2007.04813*.

[52] **Rossi, E.**, **Chamberlain, B.**, **Frasca, F.**, **Eynard, D.**, **Monti, F. and Bronstein, M.** (2020). Temporal graph networks for deep learning on dynamic graphs, *arXiv preprint arXiv:2006.10637*.

[53] **Kazemi, S.M.**, **Goel, R.**, **Jain, K.**, **Kobyzev, I.**, **Sethi, A.**, **Forsyth, P. and Poupart, P.** (2020). Representation learning for dynamic graphs: A survey, *The Journal of Machine Learning Research*, *21*(1), 2648–2720.

[54] **Huang, X.**, **Yang, Y.**, **Wang, Y.**, **Wang, C.**, **Zhang, Z.**, **Xu, J.**, **Chen, L. and Vazirgiannis, M.** (2022). Dgraph: A large-scale financial dataset for graph anomaly detection, *Advances in Neural Information Processing Systems*, *35*, 22765–22777.

[55] **Knoblauch, J.**, **Husain, H. and Diethe, T.** (2020). Optimal continual learning has perfect memory and is np-hard, *International Conference on Machine Learning*, PMLR, pp.5327–5337.

[56] **Bojchevski, A. and Günnemann, S.** (2017). Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking, *arXiv preprint arXiv:1707.03815*.

[57] **Hu, W.**, **Fey, M.**, **Zitnik, M.**, **Dong, Y.**, **Ren, H.**, **Liu, B.**, **Catasta, M. and Leskovec, J.** (2021). *Open Graph Benchmark: Datasets for Machine Learning on Graphs*, 2005.00687.

[58] **Hamilton, W.**, **Ying, Z. and Leskovec, J.** (2017). Inductive representation learning on large graphs, *Advances in neural information processing systems*, *30*.

[59] **Sen, P.**, **Namata, G.**, **Bilgic, M.**, **Getoor, L.**, **Galligher, B. and Eliassi-Rad, T.** (2008). Collective classification in network data, *AI magazine*, *29*(3), 93–93.

[60] **Kingma, D.P. and Ba, J.** (2014). Adam: A method for stochastic optimization, *arXiv preprint arXiv:1412.6980*.

APPENDICES

APPENDIX A : The Gradient Derivation for Learning Rates

APPENDIX A : The Gradient Derivation for Learning Rates

$$
\begin{aligned}
g(\alpha^j) &= \frac{\partial}{\partial \alpha^j} L_t\left(\theta_k^j\right) = \frac{\partial}{\partial \theta_k^j} L_t\left(\theta_k^j\right) \cdot \frac{\partial}{\partial \alpha^j}\left(\theta_k^j\right) \\
&= \frac{\partial}{\partial \theta_k^j} L_t\left(\theta_k^j\right) \cdot \frac{\partial}{\partial \alpha^j}\left(U\left(\theta_{k-1}^j\right)\right) \\
&= \frac{\partial}{\partial \theta_k^j} L_t\left(\theta_k^j\right) \cdot \frac{\partial}{\partial \alpha^j}\left(\theta_{k-1}^j - \alpha^j \frac{\partial \ell_t(\theta_{k-1}^j)}{\partial \theta_{k-1}^j}\right) \\
&= \frac{\partial}{\partial \theta_k^j} L_t\left(\theta_k^j\right) \cdot \left(\frac{\partial}{\partial \alpha^j}\theta_{k-1}^j - \frac{\partial}{\partial \alpha^j}\left(\alpha^j \frac{\partial \ell_t(\theta_{k-1}^j)}{\partial \theta_{k-1}^j}\right)\right) \\
&= \frac{\partial}{\partial \theta_k^j} L_t\left(\theta_k^j\right) \cdot \left(\frac{\partial}{\partial \alpha^j}\theta_{k-1}^j - \frac{\partial \ell_t(\theta_{k-1}^j)}{\partial \theta_{k-1}^j}\right) \\
&\quad \text{(Taking } \frac{\partial \ell_t\left(\theta_{k-1}^j\right)}{\partial \theta_{k-1}^j} \text{ as a constant w.r.t } \alpha^j \text{ to get the first-order MAML approximation)} \\
&= \frac{\partial}{\partial \theta_k^j} L_t\left(\theta_k^j\right) \cdot \left(\frac{\partial}{\partial \alpha^j} U\left(\theta_{k-2}^j\right) - \left(\frac{\partial \ell_t(\theta_{k-1}^j)}{\partial \theta_{k-1}^j}\right)\right) \\
&= \frac{\partial}{\partial \theta_k^j} L_t\left(\theta_k^j\right) \cdot \left(\frac{\partial}{\partial \alpha^j}\theta_0^j - \sum_{k'=0}^{k-1} \frac{\partial \ell_t(\theta_n^j)}{\partial \theta_n^j}\right) \quad (a) \\
&= \frac{\partial}{\partial \theta_k^j} L_t\left(\theta_k^j\right) \cdot \left(-\sum_{k'=0}^{k-1} \frac{\partial \ell_t(\theta_{k'}^j)}{\partial \theta_{k'}^j}\right) \quad (b)
\end{aligned}
\tag{A.1}
$$

where $U()$ represents the update function. The step (a) is obtained by the expansion and differentiation of $U()$ recursively while the step (b) is obtained under the assumption of the initial weight in the meta update stage at the time j θ_0^j is constant with respect to α^j.

CURRICULUM VITAE

Name SURNAME: Altay ÜNAL

EDUCATION:

- **B.Sc.:** 2021, Middle East Technical University, Faculty of Engineering, Department of Electrical and Electronics Engineering
- **M.Sc.:** 2024, Istanbul Technical University, Graduate School, Department of Computer Engineering

PROFESSIONAL EXPERIENCE AND REWARDS:

- 2021-ongoing Research and Teaching Assistant at Department of Computer Engineering, Istanbul Technical University

PUBLICATIONS, PRESENTATIONS AND PATENTS ON THE THESIS:

- **Unal, A.**, Akgül, A., Kandemir, M., Unal, G. (2023). Meta Continual Learning on Graphs with Experience Replay, *Transactions on Machine Learning Research*.

OTHER PUBLICATIONS, PRESENTATIONS AND PATENTS:

- Gundogdu, E., **Unal, A.**, Unal, G. (2024). A Study Regarding Machine Unlearning on Facial Attribute Data, *1st International Workshop on Responsible Face Image Processing*.