Dynamic Learning for Recommender Systems

Ion Orins

Supervised by Prof Hakan Ferhatosmanoglu

Department of Computer Science

University of Warwick

May 2022

Abstract

This dissertation aims tackle the inefficiencies of Graph Neural Network-based recommender systems by employing continual learning techniques. Recommender systems are extensively used in the technology sector with great success, with the goal of better tailoring the provided services to the individual customer's needs. An emerging recommender system paradigm based on Graph Neural Networks is becoming increasingly popular, yielding state-of-the-art results. However, it is difficult to use these algorithms in production as they need to be continuously updated to the heaps of new data that is being generated in real time and, when naïve "update" methods are used, neural network based solutions are well known to suffer of catastrophic forgetting (the tendency to perform worse on patterns in old data when the model is exclusively trained on new data). This project explores various approaches to ameliorate this issue and quantifies the trade-offs between the alternative solutions. There is a focus on testing replay- and regularisation-based methods, which are compared against naïve baseline methods. It is concluded that replay-based methods yield significant improvements both in terms of efficiency and accuracy.

Keywords: catastrophic forgetting, continual learning, graph neural networks, recommender systems, replay-based methods

Contents

List of Figures **4**

List of Algorithms **7**

List of Tables **8**

Acknowledgements **9**

1 Introduction **10**

1.1 Motivation 10

1.2 Objectives 11

1.3 The Structure of the Dissertation 12

2 Background **14**

2.1 Recommender Systems 14

2.2 Graph Neural Networks 15

2.3 Dynamic Learning 17

3 Methodology **20**

3.1 Dataset 20

3.2 Future Edge Prediction 22

3.3 Baselines 22

3.4 Evaluation Metrics 24

3.5 Continual Learning Approach 24

3.6 Framework 28

4 Results and Discussions **30**

4.1 Experimental Setting 30

4.2 Regularisation Based Approach 31

4.3 Replay Based Approach 33

4.4 Hybrid Approach 35

4.5 Exploring Different Replay Heuristics 36

4.6 Effect of Varying the Volume of Replayed Edges 37

4.7 Qualitative Analysis of Embeddings 40

4.8 Exploring Dataset Biases 42

4.9 Disabling Metapaths 45

5 Project Management **47**

5.1 Software Development Methodology 47

5.2 Resources 48

5.3 Timeline 49

5.4 Risk Assessment 50

6 Conclusions **52**

6.1 Technical Achievement 52

6.2 Limitations and Further Work 53

6.2.1 Hyperparameter Tuning 53

6.2.2 Timeframe-Related Parameter Optimisation 53

6.2.3 Exploring Other Datasets 54

6.2.4 Exploring Other Models 54

6.2.5 Exploring Other Continual Learning Methods 55

6.2.6 Improving Scalability and Usability 55

References **57**

A Accepted Command Line Arguments **71**

B Plots with the Performance of the Tested Heuristics **73**

C List of Used Metapaths **76**

D Project Specification **77**

E Progress Report **85**

F Final Presentation **90**

List of Figures

3.1 Diagram showing the relationships between the entities in the dataset 20

3.2 Example of equal number of ratings timeframes vs equal time span timeframes 21

4.1 The evolution of hit ratios over time for the baseline algorithms . 31

4.2 The effect of EWC on the performance of the model 32

4.3 The evolution of the accuracy of the model proposed in the "Streaming Graph Neural Networks via Continual Learning" [1] paper over time on the CORA [2] dataset (with EWC vs without EWC) 32

4.4 The hit ratio of the baselines and of the replay-based model over time 33

4.5 The normalised discounted cumulative gain of the baselines and of the replay-based model over time 34

4.6 The hit ratio of the baselines and of the replay-based model over time when using equal time span timeframes 34

4.7 The effect of EWC on the performance of the replay-based model 35

4.8 The evolution of the accuracy of the replay-based model proposed in the "Streaming Graph Neural Networks via Continual Learning" [1] paper over time on the CORA [2] dataset (with EWC vs without EWC) 35

4.9 The evolution of the model's hit ratio over time when using entirely random data for training (`continualfullrandom`) vs the conventional approach using the random heuristic (`continualrandom`) 37

4.10 The hit ratio of the model on timeframe 25 as a function of θ (the yellow line represents the non-continual model's accuracy when trained on all the existing data) 38

4.11 The evolution of the model's hit ratio over time for $\theta = 9$ 38

4.12 The evolution of the model's hit ratio over time for $\theta = 25$ 39

4.13 The expected speed up as a function of θ 40

4.14 The PCA of the embeddings of the vertices after training on the entire dataset (left) vs after timeframe 12 in a continual setting (right) - films are represented in red and users in blue 41

4.15 The evolution of the model's hit ratio over time when skipping the first 3 timeframes 42

4.16 The evolution of the model's hit ratio over time when the order of the ratings is shuffled 43

4.17 The evolution of the model's hit ratio over time with most metapaths disabled 46

B.1 The evolution of the model's hit ratio over time when using the *importance* heuristic 73

B.2 The evolution of the model's hit ratio over time when using the *random by importance* heuristic 74

B.3 The evolution of the model's hit ratio over time when using the *never again* heuristic 74

B.4 The evolution of the model's hit ratio over time when using the *embedding diversity* approach (program was interrupted after timeframe 15 as the performance was already clearly inferior) . . 75

List of Algorithms

1 Algorithmic overview of future edge prediction 22

2 A pseudocode summary of the implementation of the regularisation approach 25

3 Algorithmic overview of the edge importance calculation 27

4 Code extract showing the employment of parallel computing techniques when computing the embedding diversity heuristic . 27

List of Tables

3.1 The time complexities of the baseline algorithms 23

4.1 Relevant statistics about the spread of users across timeframes . 44

5.1 The timeline of the project 49

Acknowledgements

First of all, I would like to express my gratitude to my supervisor, Prof Hakan Ferhatosmanoglu for taking me under his wing after I was left with no supervisor just before the academic year started, for trusting me to tackle this complex problem and for the support and help throughout the development of this project.

I would like to thank Aparajita Haldar and Rustam Guliyev for their invaluable contributions, which brought the project to another level, for their patience to explain me all the advanced concepts that I was unfamiliar with and for sticking with me through out all the highs and lows.

I would also like to thank my friends and family members, who pushed me and kept me going when down, while enduring my continuous incoherent rambling about the ins and outs of the project.

Lastly, I am extremely grateful towards the open source community, as without their contributions the completion of the project would not have been possible in such a short amount of time.

Chapter 1

Introduction

1.1 Motivation

This project is motivated by the goal to increase the efficiency of high-accuracy recommender systems. These systems are widely used by technology businesses at scales that require enormous computational power:

1. Yelp had more than 20 million reviews submitted in the year 2020 [3];
2. Netflix has over 200 million paying subscribers [4] and well above 6,000 titles [5];
3. Youtube has exceeded 2 billion monthly active users [6].

All these companies have to continuously generate recommendations for all of their users and at this scale not much efficiency loss can be afforded. Thus, any kind of optimisation is highly relevant and can have a huge impact on

total energy consumption. For example, Alphabet, the parent company of Google, which is primarily an advertisement company and thus heavily reliant on recommender systems, is estimated to have consumed 12.4 terawatt hours in the year 2019 [7]. At the average electricity rate of 10.42 cents per kilowatt-hour [8], just a 1% decrease in consumption can save the company over $12 billion.

In addition, because this project is mainly focused on time complexity, the proposed approach can save companies precious time allowing for better scalability and more up to date recommendations on an individual level. For example, if the time complexity of an algorithm is reduced from $\Theta(n^2)$ to $\Theta(10n)$, for a user base of just 1000 users, time reductions of over 99% are far from implausible.

1.2 Objectives

The Graph Neural Network (GNN) is a relatively new architecture that yields promising results when applied to big data applications [9] and is are often considered state-of-the-art for recommendation problems. [10]

As significant amounts of data are being continuously generated, these neural networks have to be retrained in order to be able to perform well on the patterns introduced in the newer data. But because training and updating the models in order to fit them to the incoming data is an expensive computational process, investigating methods that achieve similar performance while using substantially fewer computational resource is a worthwhile endeavour.

We set out to achieve this goal by using just a subset of the available data during the aforementioned "model updating" process, hence achieving improvements in the time complexity aspect of the model. As very little to no research has been done in the indicated direction, the research prospect of this project is also motivated.

This overarching objective can be split down into and augmented with the following set of secondary objectives:

- compare and contrast different continual learning techniques;
- implement a robust framework that enables further experimentation;
- find optimal settings for the most promising approaches;
- estimate theoretical improvements and measure them practically;
- provide insights into the practicality of dynamic learning in the context of Graph Neural Networks used for recommender systems;
- identify potential paths of useful further research.

1.3 The Structure of the Dissertation

This dissertation is structured into six chapters: Introduction, Background, Methodology, Results and Discussions, Project Management and Conclusions.

The **Background** chapter familiarises the reader with building block concepts, such as recommender systems, Graph Neural Networks and dynamic

learning, including how the latter is linked to the concepts of online, incremental and continual learning. It is also suggested how these concepts tie together and motivates the reasoning behind this juxtaposition.

The **Methodology** chapter elaborates on the used dataset, evaluation metrics and baselines. It establishes the type of task performed by the model, i.e future edge prediction, and elaborates on the inner workings of the developed continual learning approach. Finally, it describes the functionalities of the implemented framework, which was used to run all the designed experiments and can potentially be utilised for further research. In short, this section explains and discuses the experimental set up.

The **Results and Discussions** chapter starts by prescribing the exact parameters used in the experimental set up and goes on presenting the results of the different tested approaches. The last three sections in this chapter go more in depth on the data analysis front and attempt to provide a more robust explanation for the obtained results.

The **Project Management** chapter talks about the employed software development methodology, the used resources (hardware, software, informational), the timeline and performs a post-completion analysis of the initial risk assessment.

The **Conclusions** begin by summarising the technical achievements and takeaways of this endeavour and ends by discussing the limitations of the project, pointing toward paths of further research that can be explored in order to overcome these limitations.

Chapter 2

Background

2.1 Recommender Systems

Recommender systems are ubiquitous nowadays, having applications in an ever increasing number of domains, such as playlist generation [11], product recommendations in stores [12] and content recommenders on social media websites [13]. The goal of a recommender system is to predict the preferences of users towards items.

One of the earliest recommender systems was developed by the GroupLens research group [14] and its purpose was to help the users of news sites find articles they were interested in more quickly. It took a collaborative filtering approach, meaning that users would rate articles and in turn they would receive recommendations for other articles which were rated high by users with similar preferences. This research group later released the MovieLens dataset [15] with the purpose of increasing the quality of movie recommandation sys-

tems. It contains a set of film ratings, as well as attributes about the rated films, and served a central role in this development of this project.

In the more recent days, recommender systems quickly gained popularity, leaving the status of niche research topic behind. As the business case of such algorithms became increasingly obvious, many corporations pushed for the development of better performing approaches. Probably the best known such incentive is the Netflix prize [16], which offered $1,000,000 to whoever managed to improve Netflix's movie recommandation system the most.

2.2 Graph Neural Networks

An exceeding number of phenomena can be modelled as graph networks and, undoubtedly, there is a tendency to use homogeneous graphs as the modelling tool. However, relationships between entities are often complex and a better approach is often the use of heterogeneous networks [17].

A powerful method compatible with this concept are Graph Neural Networks (GNNs) [18]. For example, GNN architectures, such as GraphSAGE [19], naturally incorporate the heterogeneity of the graph by taking into account the type of connection when passing information between two vertices. GNNs have been proven to be an exceedingly useful neural network architecture with applications in recommender systems [20], natural language processing [21], drug discovery [22], etc.

The concept of a Graph Neural Networks was first introduced into the literature by Scarselli et al. [23] in 2009 and can be understood as a generalisa-

tion of Convolutional Neural Networks [24], as their input can be interpreted as a 2D/3D lattice lattice-like graph, but also as a generalisation of Recurent Neural Networks [25], by removing the constraint that edges have to go forwards through time.

The premise of Graph Neural Networks is that the input is a graph structure and the outputs are representations of some kind of either vertices, edges or the entire graph structure. These representation are further used to perform an ordinary task such as classification or regression. [26]

A fairly popular class of Graph Neural Network models is the message-passing neural network (MPNN). [27] This model begins by assigning all vertices hidden states, called vector representations. Then, these vector representations are updated based on an aggregation of the vector representations of neighbouring vertices. Through this mechanism, information is passed from one vertex to another, hence the term "message-passing".

Moreover, it has been shown that an effective method to improve GNNs' performance is to augment them with metapath awareness, achieving state-of-the-art performance on recommender systems [10]. Metapaths are "sequences of relations defined between different object types" [28], and by considering them the neural network can be aware of the deeper semantics of the relationship between two nodes further apart in the network.

This project builds on concepts presented in the "Metapath- and Entity-aware Graph Neural Network for Recommendation" (PEAGNN) paper [10], which provides the theoretical framework for the neural network architecture to be improved and an example implementation which was used as the basis

for this project. It compiles several architectures of message-passing neural networks and offers augmentations through the aforemantioned metapath awareness technique.

Out of all the reference implementations provided by the PEAGNN repository, the model based on graph attention networks [29] (PEAGAT) has been chosen to be worked on, as it is currently the state-of-the-art for the chosen dataset. The idiosyncrasy of this architecture is the use of an attention mechanism as the aggregator function.

This model, in order to make a recommendation, runs the vector representations of the vertices through the graph convolution layers in order to obtain the embedding for each entity. Then it concatenates the embedding of a user vertex with the embedding of an item vertex and runs the result through a fully connected neural network. [10] A high value output represents a greater likelihood of a match, while a low value represents a lesser likelihood of a match.

2.3 Dynamic Learning

Dynamic learning here is used as an umbrella term for any kind of technique which enable machine learning models to perform better in a dynamic environment. A more rigorous term is *incremental learning*, which is defined as continually using input data to extend the model's existing knowledge. [30] The reason incremental learning is useful is because often new data becomes available in a continuous fashion after the deployment of the machine learning

model, in which context the posed problem is referred to using the term *online learning* [31].

However, blindly training the model on the new data, while ignoring the old data, the phenomenon of *catastrophic forgetting* [32] might emerge, particularly when dealing with connectionist models. This term was coined to describe the tendency of a gradient-based neural network's performance on a task to very quickly degrade once trained on a new task.

Recently researchers have started to develop methods to overcome this issue by working on a number of techniques, commonly referred to as *continual learning* [33]. Thus, *continual learning* encompasses all methods that allow models to learn an increasing amount of tasks without a significant hit on the performance of previously learned tasks. These methods can be split into three categories: regularisation-based methods, parameter isolation methods and replay methods.

Regularisation-based methods work by adding terms to the loss function that attempt to penalise the neural network when parameter updates decrease the performance on previous tasks. One example of a regularisation method is called *elastic weight consolidation* (EWC) and it was introduced in the paper "Overcoming catastrophic forgetting in neural networks" [34]. The basis of this method is to add a term to the loss function which penalises the model when the parameters are updated to values that are significantly different to the ones resulting from training on previous tasks, thus slowing down learning on the weights important for the old tasks.

Replay-based methods are perhaps the most straightforward: they either

attempt to identify important data points from previous tasks and use them alongside the new data points when training for the new task [35] or generate new synthetic training data that helps the neural network "remember" how to perform previously learned tasks [36].

Parameter isolation methods are based on the idea that different parts of a neural network can learn distinct tasks. One proposed approach is starting with a simpler architecture and progressively adding new features such as neurons or layers, whose purpose is to learn the new task. [37]

This project mainly focuses on replay methods, while slightly touching on regularisation methods. Parameter isolation methods are ignored due to the necessity to limit the scope of this project, as these methods greatly increase the architectural complexity of an already quite complex GNN and are usually rather used in reinforcement learning contexts [37]. On the other hand, regularisation techniques tend to have minimal impact on performance, while replay methods were used successfully in a wide range of applications [35, 1, 38].

The paper "Streaming Graph Neural Networks via Continual Learning" [1] was used as a reference for implementing continual learning in the context of GNNs, as it describes a method that combines a regularisation-based method and a replay-based method to solve a classification problem on a streaming network dataset with not much accuracy loss compared to retraining the network from scratch.

Chapter 3

Methodology

3.1 Dataset

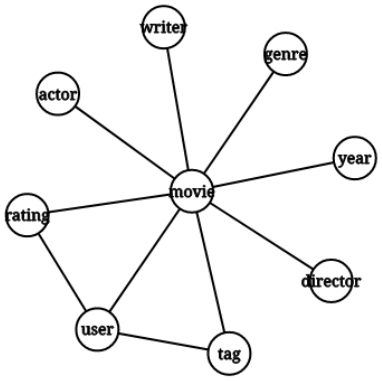


Figure 3.1: Diagram showing the relationships between the entities in the dataset

As mentioned in the background chapter, the MovieLens dataset was used as a basis for measuring the performance of the developed algorithm. It consists of ratings on a scale of 1 to 5 given by users to films, tags applied by users

to films (such as classic, comedy or sci-fi) and the following film attributes: actors, writers, genre, year, director (see figure 3.1). The `ml-latest-small` variant of the dataset was used, which consists out of 100,000 ratings and 3,600 tag applications applied to 9,000 movies by 600 users. This dataset can be easily modelled as a graph by treating films, users, tags, actors, writers, genres, years and directors as vertices and ratings, taggings, acting, writing, belonging to a genre, being released in a year and directing as edges.

However, this dataset is static in nature and in order to apply continual learning techniques, it first needed to be transformed into a dynamic dataset. By using the timestamp attribute of the rating entries, the dataset could be split into batches, of either equal number of ratings or equal time span (see figure 3.2). The author from here on will refer to these batches using the term "timeframes".

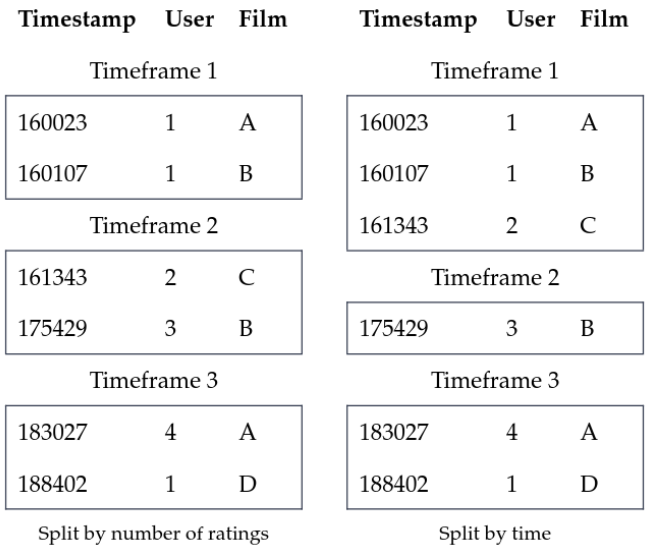


Figure 3.2: Example of equal number of ratings timeframes vs equal time span timeframes

3.2 Future Edge Prediction

In order to measure the performance of the developed system, a useful and well defined task had to be devised. A straight forward task to perform for the algorithm that is also relevant in the context of film recommendations is future edge prediction. In simpler terms, given all the ratings up to timestamp t, predict what films the users will watch starting with timestamp $t + 1$. For the general case of timestamp n, algorithm 1 shows a high level overview of the logic.

Algorithm 1 Algorithmic overview of future edge prediction

1: Let E_n be the set of all ratings from timeframe n and $E_{1\text{to}n}$ be the set of all ratings from timeframes 1 to n inclusive
2: add edges E_n to the internal graph representation of the GNN
3: select a set of edges $E \subset E_{1\text{to}n}$
4: train on set E
5: evaluate on set E_{n+1}

3.3 Baselines

In order to prove that the approach proposed in this dissertation is indeed an improvement over more traditional or naïve methods, but also to quantify this improvement, several baseline algorithms have been implemented. These are described in more detail in the following paragraphs.

Retrained [1] is the most computationally intensive, but at the same time it is expected to perform the best. The approach is simple: retrain model from scratch on all available data, i.e. on the data from the current timeframe and

all the past ones.

Pretrained [1] lies on the opposite end of the spectrum, as it is least computationally intensive but expected to perform the worst. In this scenario, the model is only trained on the data from the first timeframe and just reused in the other timeframes.

Single [1] is a middle ground from a computational perspective: the model is trained from scratch only using data from the current timeframe.

Online [1] is extremely similar to the *single* approach, but it introduces an *online learning* aspect for the first time. It uses the same training schedule and data as the previous baseline, but the model is fine-tuned on the new data, rather than trained from scratch (i.e. the parameters of the model are retained from the end of one timeframe to the begging of the next one, rather than reset).

Table 3.1 shows the complexities of the baselines with respect to the input data (e = average numer of recommendations per timeframe), both in the case of a single timeframe and in total, across the entire dataset.

	Per Timeframe	**Total**
Retrained	$\mathcal{O}(e)$	$\mathcal{O}(e^2)$
Pretrained	$\mathcal{O}(1)$	$\mathcal{O}(e)$
Online	$\mathcal{O}(e)$	$\mathcal{O}(e)$
Single	$\mathcal{O}(e)$	$\mathcal{O}(e)$

Table 3.1: The time complexities of the baseline algorithms

3.4 Evaluation Metrics

Two evaluation metrics were used in order to measure the performance of the different approaches: hit ratio and normalised discounted cumulative gain. Hit Ratio at k (HR@k) is defined as the number of hits (i.e. the correct prediction is in top k most confident guesses returned by the model) divided by the number of users in the test set. Normalised discounted cumulative gain (nDCG) is a metric that also takes into account the ordering of the prediction and is defined as the ratio between the discounted cumulative gain (DCG) and the ideal discounted cumulative gain (IDGC), i.e. the maximum possible DCG in the given situation. The formulas are as follows:

$$nDCG@k = \frac{DCG@k}{IDCG@k}$$

$$DCG@k = \sum_{i=1}^{k} \frac{rel_i}{\log_2(i+1)}$$

where $rel_i = 1$ if the guess at position i is correct or 0 otherwise.

3.5 Continual Learning Approach

The first tested approach is *elastic weight consolidation*, a regularisation based approach, which adds the following term to the loss function [34]:

$$\sum_i \frac{\lambda}{2} F_i (w_i - w^*_{prev,i})^2$$

where λ sets the importance of the previous timeframe compared to the current one, i labels each parameter, F is the Fisher information matrix [39], w the current parameters of the model and w^*_{prev} the parameters of the model after being trained on the previous timeframe. This modification to the loss function was done to the *online* baseline in order to get a fist continual GNN model. An overview of the implementation in pseudocode is shown in algorithm 2.

Algorithm 2 A pseudocode summary of the implementation of the regularisation approach

for each: $timeframe \in timeframes$

1: $model.add\_edges(timeframe.ratings)$ where the used loss function is $model.loss\_function\_for\_current\_task(model.current\_parameters) + model.ewc\_function(model.current\_parameters - model.saved\_parameters)$

2: $model.save\_parameters()$

The second tested approach is a replay-based method and works by having the construction of the set of edges used for training in each timeframe $E = E_n \cup \{e | e \in E_{1 \text{to} n-1} \wedge s(e)\}$, where $s(e)$ is true if e is in the set of size $(\theta - 1)|E_n|$ containing the edges which output the highest value when used as input in a heuristic function h. I would like bring attention to the θ hyperparameter as it is going to be mentioned quite often in the Results and Discussion chapter: it can be seen as a control for the number of data points used for replay.

The following heuristic functions were devised and tested:

1. **random**: $h(e) =$ random number;
2. **importance**: $h(e) = importance(e)$;
3. **sliding window**: $h(e) =$ random positive number if $e \in E_{n-k \text{to} n}$ else 0, where k is a hyperparameter;

4. **never use again**: $h(e) =$ random positive number if e has not been chosen by this heuristic before else 0;

5. **random by age**: $h(e) =$ random number proportional to $2^{age(e)}$;

6. **random by importance**: $h(e) =$ random number proportional to $importance(e)$;

7. **softmax**: $h(e) =$ random number proportional to $softmax\left(\frac{importance(e)*b^{age(e)}}{\tau} + \epsilon\right)$, where b, τ and ϵ are hyperparameters.

In addition, a mixed approach was tested, dubbed **mixed random**, which chooses a proportion α of the edges using the **random by importance** heuristic and a proportion $1 - \alpha$ of the edges using the **random by age** heuristic.

The last tested approach, named **embedding diversity**, uses the heuristic function $h(e) = \sum_{e' \in S} dist(e, e')$ where $S = \{$ already selected edges $\}$ and $dist$ is the euclidean distance between the two points. However, this heuristic was used to construct the entire training set E, starting with a set containing a randomly selected edge, rather than the set E_n.

Following, the functions $age(e)$ and $importance(e)$ will be defined. The age of an edge is simply defined as $age(e) = n - k - 1$ s.t. $e \in E_k$. Importance is a more convoluted and perhaps more arbitrary metric. For the purposes of this project, importance is defined as the amount of change in the embeddings of the edge before and after updating the topology of the GNN (step 1 in algorithm 1). [1] To make it more clear, the steps taken to calculate this importance are represented in algorithm 3.

Algorithm 3 Algorithmic overview of the edge importance calculation

Let $e = (u, v)$ and $emb(u)$ returns the current embedding of vertex u
$\text{emb}_0 \leftarrow concat(emb(u), emb(v))$
update the topology of the graph
$\text{emb}_1 \leftarrow concat(emb(u), emb(v))$
$\text{imp}_e \leftarrow ||\text{emb}_1 - \text{emb}_0||$

Time complexity-wise, if we assume the heuristic function can be computed in $\mathcal{O}(1)$, the continual learning approach fits in the same category as the single and online baselines. In practice, even if this might not be the case, hash tables and parallel computing methods were employed in order to make the computation insignificant relative to the entire training process. An example of such optimisation is shown in algorithm 4.

Algorithm 4 Code extract showing the employment of parallel computing techniques when computing the embedding diversity heuristic

```
distances = torch.cdist(edge_embs, edge_embs)

selected_indices = []
distances_to_se = torch.zeros(len(edge_embs)).to('cuda')

for _ in range(no_samples):
    index = np.random.randint(no_samples)

    if len(selected_indices) > 0:
        distances_to_se += distances[:, selected_indices[-1]]
        index = torch.argmax(distances_to_se).item()

    selected_indeces.append(index)
```

3.6 Framework

As many approaches were tested, and all if them had many hyperparameters that could be tweaked, in order to make experimentation easier, a big part of this project was implementing a robust framework. This allowed the author to run numerous tests with minimal changes to the code. Implemented features of this framework that proved to be useful include:

- timeframe manipulation (helped the exploration of the effects of the size and the nature of the timeframes on the performance and also the testing of different hypothesis regarding biases in the dataset):
  - changing the size of the timeframes;
  - switching between equal time span and equal number of ratings type timeframes;
  - skipping timeframes;
  - changing the ordering of the ratings (by timestamp/random);
- heuristic function manipulation (numerous hypothesis regarding optimal data selection strategies constantly arose and this allowed for a quickfire fashion of testing them):
  - changing the definition of the heuristic function;
  - changing the definition of the importance function;
  - changing the used distance metric;

- investigation tools (to help visualise the effects of the made changes on the results):
  - performing principal component analysis on the vertex embeddings;
  - displaying the evolution of the used evaluation metrics over time;
  - displaying the selected edge distribution w.r.t. their original timeframe (i.e. n s.t. $e \in E_n$);
- model tweaking (helped the exploration of different common deep learning techniques on the performance of the network):
  - disabling metapaths selectively;
  - adding a batch normalisation layer;
  - using one hot encoding as vertex representations;
- continual learning parameter tweaking (helped the exploration of trade-offs between speed and accuracy):
  - changing the λ coefficient in the context of the EWC approach;
  - changing the size of the set of selected edges (the parameter θ);
- running multiple instances of the model at the same time;
- switching the testing set between future edges and already present edges.

A full list of the command line arguments accepted by the program, accompanied by default values and an explanation, can be found in the appendices.

Chapter 4

Results and Discussions

4.1 Experimental Setting

The MovieLens dataset was split into 25 timeframes containing aproximately 3000 ratings each. These figures were chosen in order to create a challenging setting but at the same time keep the problem tractable.

For evaluation, the hit ratio at 10 (HR@10) metric is used mainly, but occasionally the normalised discounted cumulative gain at 10 (nDCG@10) is provided in order to prove the robustness of the solution.

When not specified to be a different value, the θ parameter of the replay-based methodology is set to the value 1.5. Similarly, unless specified otherwise, equal number of ratings timeframes are used and the *random* heuristic is employed.

Most plots will show the evolution of the tested approach's performance throughout the first 24 timeframes (as the 25^{th} timeframe does not have a fol-

low up in order to perform future edge prediction).

For example, figure 4.1 shows the hit ratios over time of the four baseline algorithms. It can be seen that as expected, the *retrained* baseline performs significantly better than all the others and that *online* performs marginally better than *single*. Additionally, the plot shows the *catastrophic forgetting* phenomenon, as the hit ratio of the *online* baseline drops sharply after a few timeframes. Maybe unexpectedly, the *pretrained* baseline performs slightly better than the *single* and *online* baselines, at least in the short run, hinting at a dataset bias that will be discussed further down in this document.

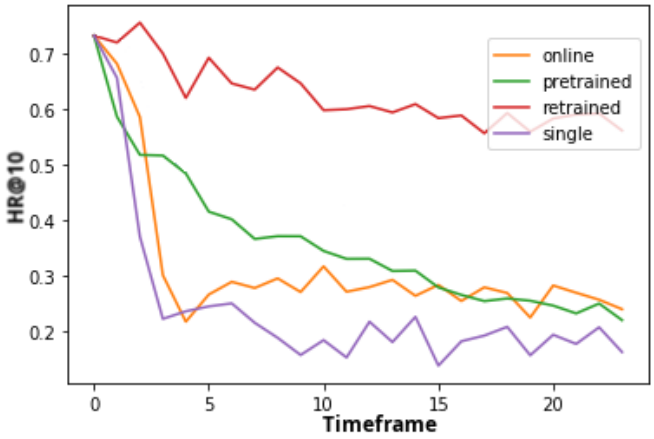


Figure 4.1: The evolution of hit ratios over time for the baseline algorithms

4.2 Regularisation Based Approach

Elastic weight consolidation, the tested regularisation-based method had no impact whatsoever on the performance of the network, as seen in figure 4.2. Multiple values for the λ parameter were tried, ranging from 50 to 10^5, all yielding comparable results.

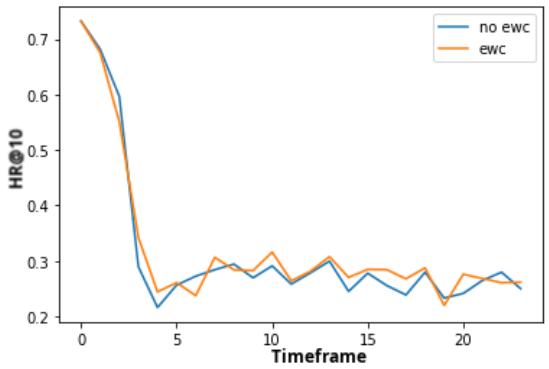


Figure 4.2: The effect of EWC on the performance of the model

As a sanity check, we tried running the code from the paper "Streaming Graph Neural Networks via Continual Learning" [1] with and without elastic weight consolidation and the results are similar to our findings (figure 4.3).

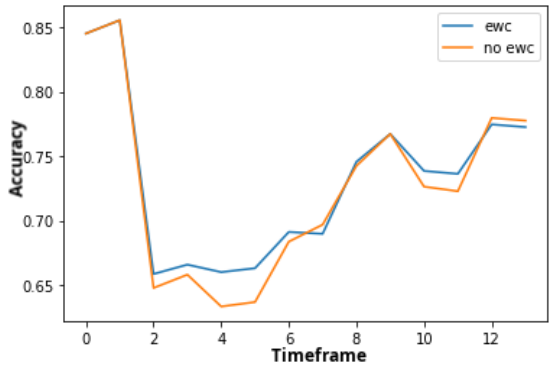


Figure 4.3: The evolution of the accuracy of the model proposed in the "Streaming Graph Neural Networks via Continual Learning" [1] paper over time on the CORA [2] dataset (with EWC vs without EWC)

The lack of improvements resulting from the use of elastic weight consolidation in the context of GNNs was also previously found by the authors of the paper "Catastrophic Forgetting in Deep Graph Networks: an Introductory Benchmark for Graph Classification" [40], who argue that the impracticability

of this method is a result of the heavy reliance of the model on the values of the vector representations of the vertices, thus constraints on the parameters of the hidden layers being of little use.

4.3 Replay Based Approach

By setting the parameter $\theta = 1.5$ and using the random heuristic it can be seen in figures 4.4 and 4.5 that the replay-based method outperforms four out of the five baselines, but still trailing significantly behind the *retrained* baseline. The value chosen for θ is just above one in order to highlight the advantages of method while at the same time keeping the run time very close to the *single* and *online* baselines.

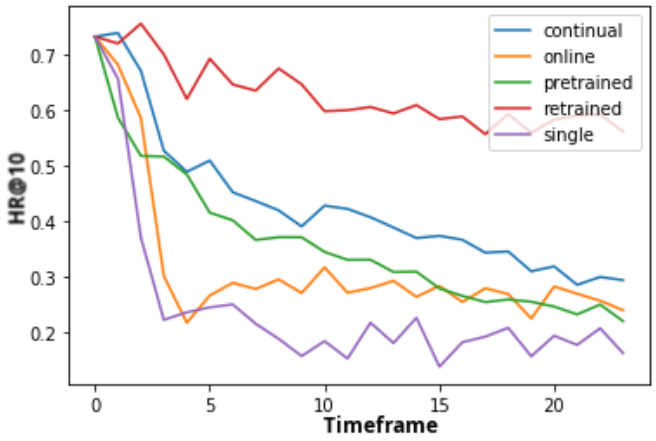


Figure 4.4: The hit ratio of the baselines and of the replay-based model over time

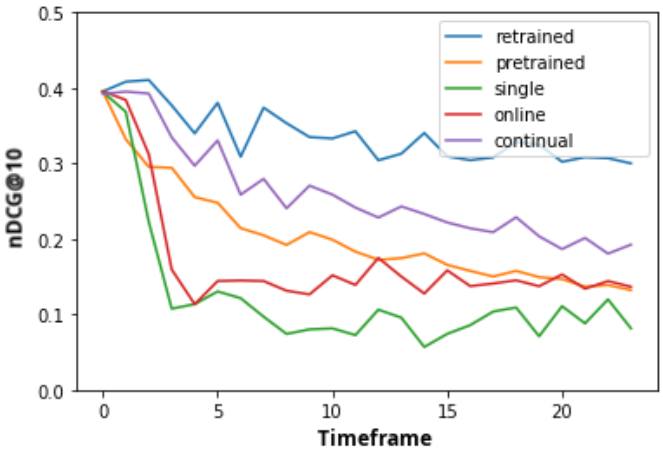


Figure 4.5: The normalised discounted cumulative gain of the baselines and of the replay-based model over time

When using equal time span timeframes, the results are similar, as seen in figure 4.6).

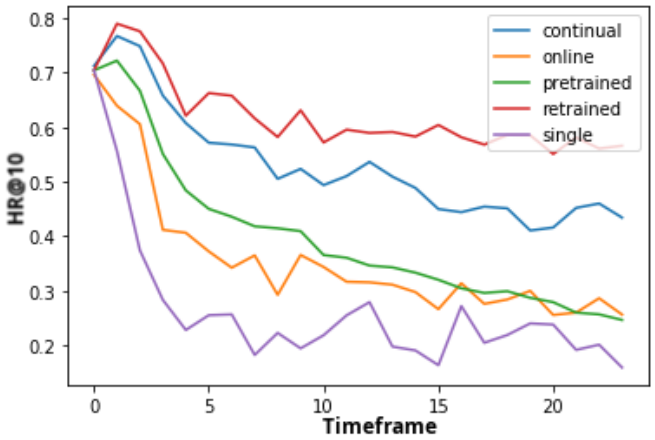


Figure 4.6: The hit ratio of the baselines and of the replay-based model over time when using equal time span timeframes

4.4 Hybrid Approach

As the paper "Streaming Graph Neural Networks via Continual Learning" employs both a regularisation and a replay-based approach at the same time, it has been hypothesised that the interaction between the two might be a crucial factor in boosting the accuracy of the solution. However, running tests with both regularisation and replay enabled shows little to no improvement over using just the replay method alone in either our (figure 4.7) or their (figure 4.8) solution.

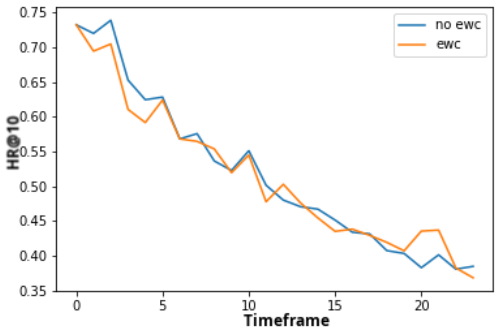


Figure 4.7: The effect of EWC on the performance of the replay-based model

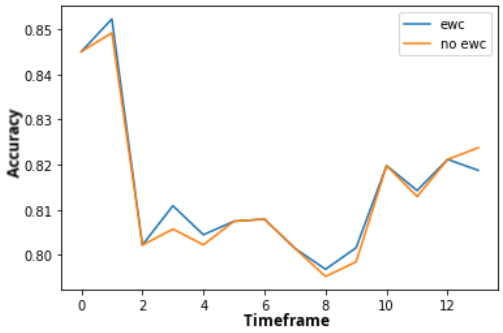


Figure 4.8: The evolution of the accuracy of the replay-based model proposed in the "Streaming Graph Neural Networks via Continual Learning" [1] paper over time on the CORA [2] dataset (with EWC vs without EWC)

4.5 Exploring Different Replay Heuristics

The performance of the different heuristics can be classified relative to the random heuristic in three groups, as follows:

- **no better than random** (these heuristics are usually worse but by tweaking the parameters, they become more or less the same as random): sliding window (when k=25), softmax (when τ and ϵ are big and b is small), random by age, mixed random (when $\alpha = 1$, i.e. the heuristic is identical to the age heuristic);
- **worse than random** (consistently underperforms): importance, random by importance, never use again;
- **significantly worse than random** (>50% drop in performance): embedding diversity.

Explanations for the performance of these different heuristics will be discussed in the section Exploring Dataset Biases and comprehensive plots showing the performance of each approach can be found in the appendices.

Following the discovery of the lack of utility brought by the more advanced heuristics, the approach of training on the edges from the current timeframe plus selected edges from the previous timeframes approach was questioned. Hence a new methodology was designed, which chooses all edges for training randomly across all available timeframes, without differentiating between the current and the past timeframes. The results were not much different to

the ones obtained using conventional approach combined with the random heuristic (figure 4.9), showing that feeding the model all the available data throughout its lifetime is not as important as having a representativ selection of edges.

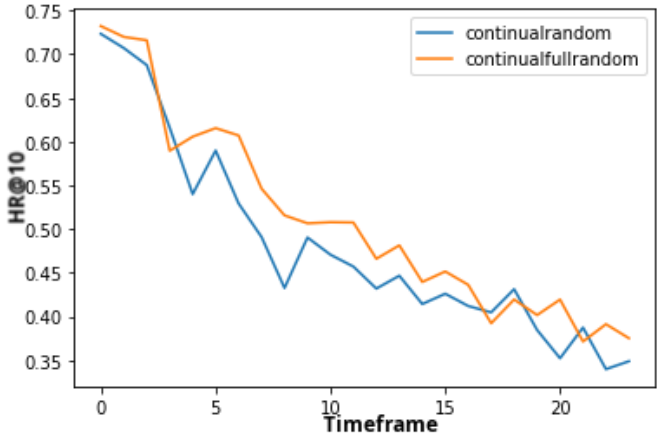


Figure 4.9: The evolution of the model's hit ratio over time when using entirely random data for training (`continualfullrandom`) vs the conventional approach using the random heuristic (`continualrandom`)

4.6 Effect of Varying the Volume of Replayed Edges

This section explores the effects of varying the amount of data used for replay, i.e. tweaking the θ parameter. As seen in figure 4.10, by raising the amount of data used for replay, the performance of the network increases in a logarithmic fashion and once θ is equal to 9, its performance (HR@10=0.56) beats the performance of the retrained baseline (HR@10=0.55). To ensure that this gain in performance is consistent across all timeframes, figure 4.11 shows the evolution of the $\theta = 9$ variant across the entire dataset.

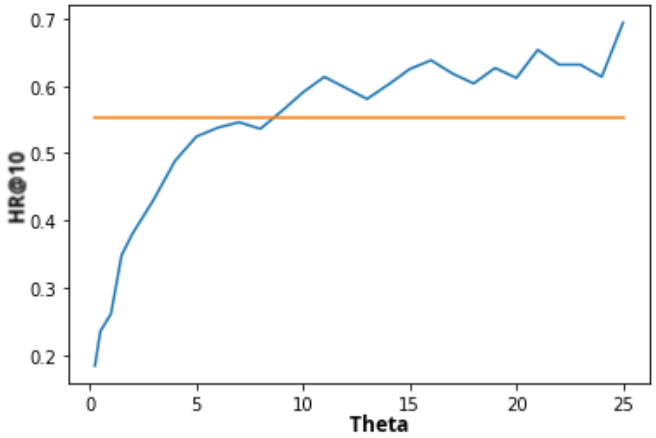


Figure 4.10: The hit ratio of the model on timeframe 25 as a function of θ (the yellow line represents the non-continual model's accuracy when trained on all the existing data)

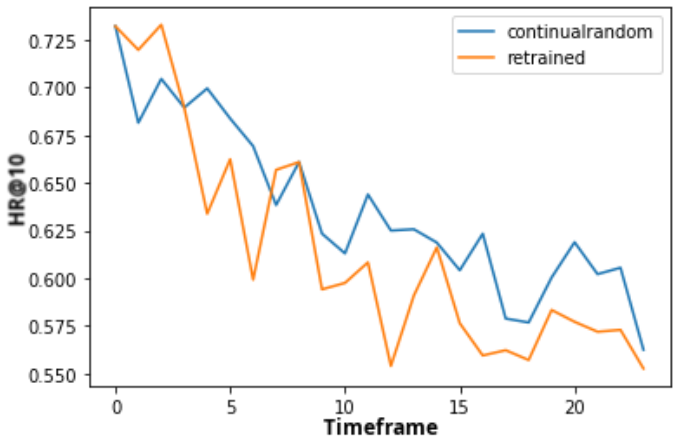


Figure 4.11: The evolution of the model's hit ratio over time for $\theta = 9$

It can be also be noticed that when θ is 25, the continual model (HR@10=0.69) beats the baseline retrained model by more than 25% (figure 4.12).

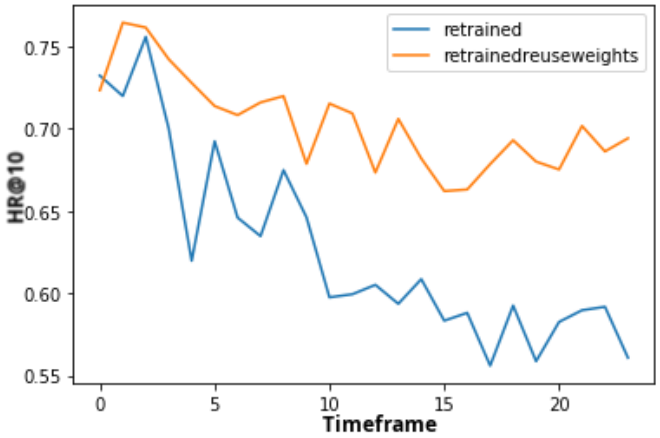


Figure 4.12: The evolution of the model's hit ratio over time for $\theta = 25$

Now that it has been shown that a continual approach can achieve similar results to the naïve best case scenario represented by the retrained baseline, it needs to be shown that the expected boost in time efficiency is indeed present. Assuming no overheads caused by the computation of the heuristic function and a linear correlation between the number of data points used for training and the time taken by the training process, we arrive to the formula below, which provides a good estimate of the expected speed-up as a function of θ, when considering the sum of training times on all the timeframes:

$$speedup(\theta) = \frac{\sum_{i=0}^{tf} i}{\sum_{i=0}^{tf} min(i, \theta)}$$

where tf is the number of timeframes the dataset was split into. Figure 4.13 shows the expected speed-ups for different values of θ.

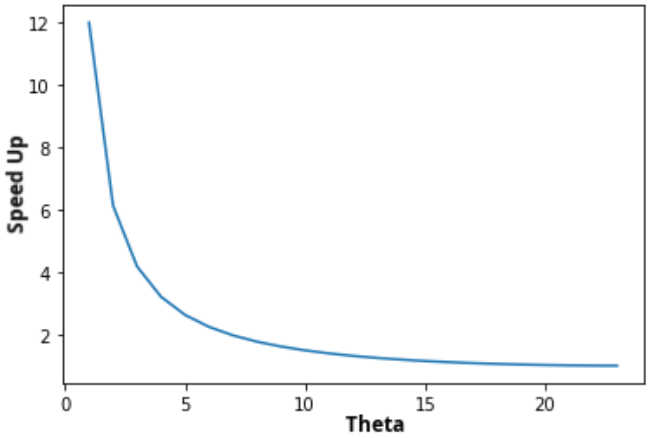


Figure 4.13: The expected speed up as a function of θ

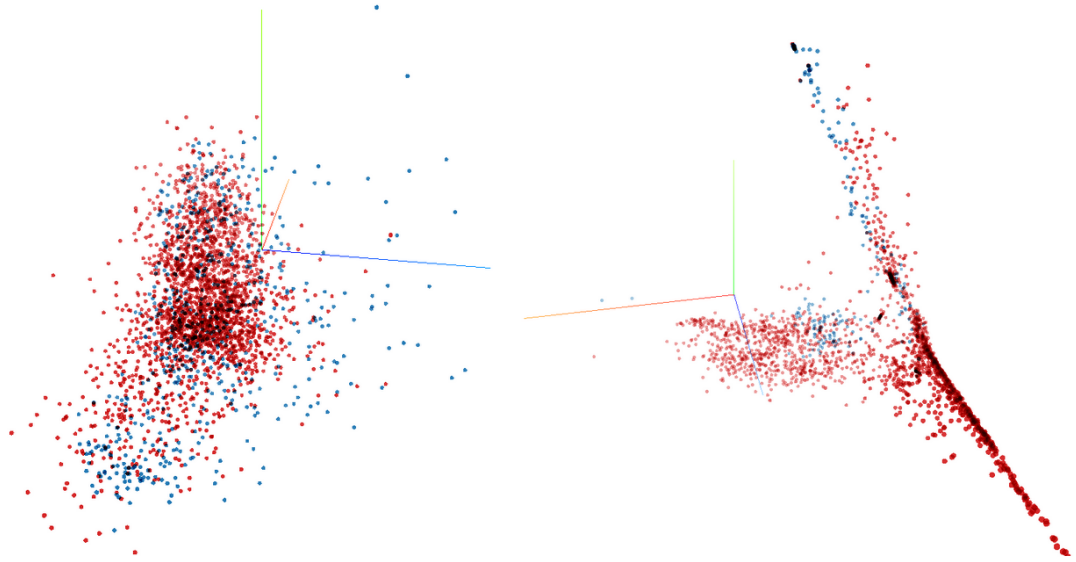
For the case when $\theta = 9$, the expected speed-up is 1.61 and the actual measured speed-up is 1.38 (the time was reduced from 131 minutes to 95 minutes). However, it is reasonable to say that slightly better code optimisation could bring the practical speed-up to a value closer to the theoretical one.

Hence, the proposed approach is both better and faster than the best performing baseline scenario. At the same time, it offers the possibility to trade speed for performance by tweaking the θ parameter.

4.7 Qualitative Analysis of Embeddings

An interesting issue to look into is the distribution of the embeddings assigned by the GNN to the vertices. When plotting these in 3D space, by using principal component analysis, it was observed that continual methods result in a rather an abnormal spread compared to the original non-continual model (figure 4.14). It is not conclusive whether this affects the performance of the GNN

in any way. Nonetheless, two methods were investigated that had the potential to lead to a more conventional spread in the embedded space: adding a batch normalisation layer and one hot encoding the vector representations of the vertices.



Embeddings after conventional training

Embeddings after timeframe 12

Figure 4.14: The PCA of the embeddings of the vertices after training on the entire dataset (left) vs after timeframe 12 in a continual setting (right) - films are represented in red and users in blue

Batch normalisation [41] re-scales and re-centres the inputs of the next layer in the neural network, hence it was reasonable to believe that it could encourage the embeddings to spread in a manner closed to a normal distribution. However, the result was that is significantly hurt the performance of the model.

Another tried approach is one hot encoding the vector representations of the vertices, because by fixing these in place, it disables feature training and should offer a greater stability to the distribution of the embeddings. It mar-

ginally boosted the accuracy of the model, however it is reasonable to say that this rise came from the increase of the dimensionality of the embeddings as a higher number of dimensions was required to one hot encode the entire dataset (from 64 dimensions to 2902).

4.8 Exploring Dataset Biases

As discussed in the Methodology chapter, the pretrained baseline performing better than the single and online baselines hints that the timeframes at the beginning contain more information than the later ones and have a greater importance when used for training. In order to test this hypothesis, we ran the continual model on the dataset with the first 3 timeframes skipped (figure 4.15) and the results became much more chaotic, confirming the existence of the bias.

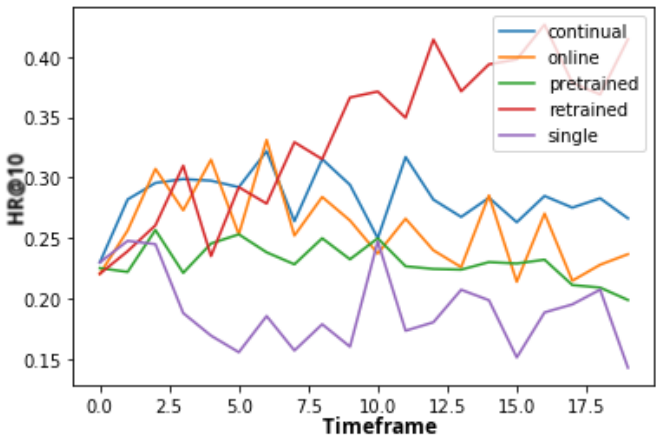


Figure 4.15: The evolution of the model's hit ratio over time when skipping the first 3 timeframes

In order to ensure that the proposed approach is not reliant on this bias,

we ran the test on a shuffled dataset, which guaranteed that no timeframe is more relevant than another. The plot looks strikingly different (figure 4.16), but nevertheless confirms the superiority of the continual approach. At the same time, the pretrained baseline now performs worse than online and single, as one would expect.

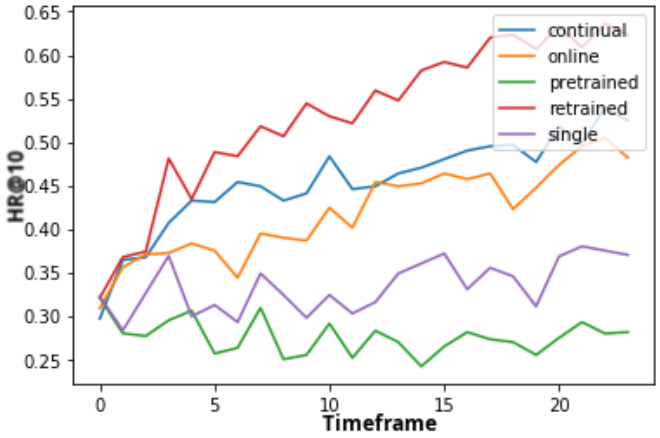


Figure 4.16: The evolution of the model's hit ratio over time when the order of the ratings is shuffled

In order to find out what results in this bias, we decided to look into the distribution of the rating counts with respect to the users that wrote the reviews within each timeframe. The table 4.1 shows that earlier timeframes have a higher mean review counts, while the user counts and the variance in review counts are lower. Thus, it can be concluded that by having a higher topological diversity, training on earlier timeframes results in a higher accuracy.

Timeframe	User Count	Mean Rating Count per User	Standard Deviation of Rating Count per User
1	55	54.55	26.63
2	55	54.55	39.15
3	44	68.18	66.75
4	29	103.45	101.18
5	18	166.67	158.06
6	31	96.77	105.44
7	39	76.92	88.71
8	29	103.45	107.94
9	31	96.77	136.90
10	28	107.14	108.54
11	36	83.33	132.79
12	40	75.00	112.99
13	33	90.91	152.89
14	34	88.24	150.04
15	35	85.71	91.41
16	40	75.00	125.48
17	42	71.43	109.72
18	54	55.56	75.02
19	41	73.17	99.54
20	44	68.18	64.19
21	34	88.24	138.20
22	35	85.71	129.39
23	34	88.24	120.10
24	34	88.24	162.11
25	37	81.08	189.60

Table 4.1: Relevant statistics about the spread of users across timeframes

This sheds a light on the performance of the tested heuristics. The heuristics from the worse than random group impose constraints that limit diversity thus hurting performance. The reason this happens when using the importance-based heuristics is a more complex story: the more important edges are indeed identified, but by training on these edges, it boosts their importance more, creating feedback loop that keeps selecting the same edges over and over again and ultimately bringing accuracy down.

As a takeaway from this section the notion of edge importance becomes more and more meaningless if one considers them in isolation, and it is obvious that the relationships between edges is highly relevant. Hence it is difficult to define importance and the goal should shift towards defining representative sets of edges. Moreover, the fact that the *embedding diversity* performs so poorly shows that diversity in the topological space is more relevant than diversity in the embedded space.

4.9 Disabling Metapaths

In order to see the effects of metapaths, the model was run with seven out of nine metapaths disabled, only leaving the user-movie-user and movie-user-movie metapaths enabled. A complete list of all the existing metapaths can be found in the appendices. The results are plotted in figure 4.17 and it can be seen that the GNN did not take an observable hit on its performance. On the one hand, it shows that most metapaths only have a small contribution to the accuracy of the model. On the other hand, it demonstrates that the approach is robust and it is not dependant on the presence of these features.

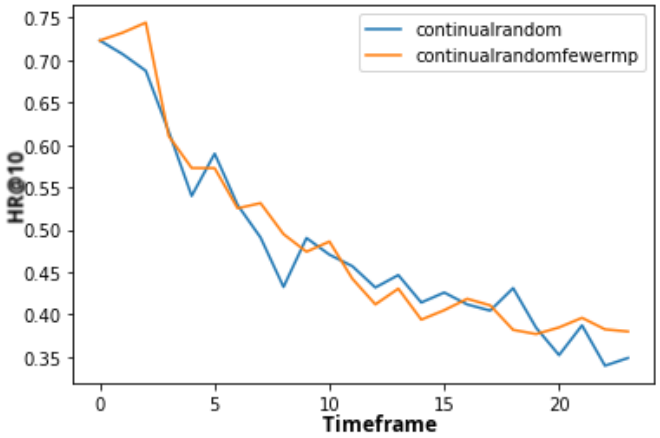


Figure 4.17: The evolution of the model's hit ratio over time with most metapaths disabled

Chapter 5

Project Management

5.1 Software Development Methodology

As this project falls under the research category but also has a software development component, its complexity prohibited a plan-driven approach, Agile's increased flexibility being optimal. There were many techniques to test, plenty of potential points of failure, and a considerable number of possible routes to be explored, hence, the Kanban methodology fit best the roadmap of the project. This allowed for quickly updating the backlog of tasks (adding new ideas and removing tasks that became obsolete), pausing and switching between diverging approaches, as their versatility was being discovered and dead-ends were reached.

The testing component of the project was done using publicly available datasets, following the well known train - test split methodology.

It is also important to mention that a GitHub repository was being used for

version control and weekly meetings were conducted with PhD students working on related projects under the supervision of Prof Hakan Ferhatosmanoglu in order to gather feedback on the progress, further actions and better familiarise with the technical concepts applied in this project.

5.2 Resources

Hardware-wise, a personal laptop was used to write the documentation, create plots and diagrams, implement short codebases intended as proof of concepts, and also to interface with DCS machines, on which the main bulk of work was done.

The main programming language used is Python, with the help of various machine learning and data science libraries (e.g. PyTorch [42], scikit-learn [43], pandas [44], NumPy [45]). In addition, the parallel computing library CUDA [46] was employed to accelerate the training and evaluation of the neural networks by running intensive computations on discrete GPUs. Occasionally, short automation scripts were written using the shell language Bash. Multiple datasets and public repositories with open-source implementations of concepts that were used in this project are available online, which helped accelerate the code writing process. More specifically, the following resources were used:

- ContinualGNN [1] - which offers an implementation of continual learning on Graph Neural Networks;
- PEAGNN [10] - which offers implementations of multiple architectures of state-of-the-art Graph Neural Networks for recommender systems;

- MovieLens [15] - a dataset that contains film ratings, as well as attributes about both users and the rated films.

5.3 Timeline

T1	W-2	supervisor meeting, agree on topic, familiarise with concepts
	W-1	familiarise with concepts
	W0	first weekly meeting within the research group, familiarise with concepts
	W1	familiarise with concepts
	W2	hand-in project specification, familiarise with concepts
	W3	familiarise with libraries, public repositories and datasets
	W4	familiarise with libraries, public repositories and datasets
	W5	supervisor meeting, familiarise with libraries, public repositories and datasets
	W6	implement framework
	W7	implement framework
	W8	implement regularisation method
	W9	hand-in progress report, supervisor meeting, implement replay method
	W10	implement replay method
Winter Break		test different timeframe types and sizes
T2	W1	test different importance heuristics
	W2	test different importance heuristics
	W3	test different importance heuristics
	W4	supervisor meeting, test different importance heuristics
	W5	investigate the distribution of edge selections by different heuristics
	W6	supervisor meeting, investigate the effect of changing the theta parameter
	W7	supervisor meeting, investigate different heuristics
	W8	supervisor meeting, investigate different heuristics
	W9	final presentation, investigate dataset biases
	W10	investigate dataset biases
Spring Break		run final tests, wrap-ups, document code, write final report
T3	W1	write final report
T3	W2	final report submission

Table 5.1: The timeline of the project

5.4 Risk Assessment

At the beginning of the project, the following risks have been identified, with their corresponding contingency / mitigation plans:

- **R1**: As frameworks and public repositories with a high number of dependencies will be used, there is a great chance that software incompatibilities may arise. In order to mitigate this, isolation techniques such as virtual environments, containers and virtual machines would be used.
- **R2**: Because the risk of data loss is present, just like in any software project, backups would be used and data would be stored both locally and in the cloud.
- **R3**: A hardware failure that restricts access to development may occur. As data is backed up, there is always an alternative to resume development: personal laptop, DCS machines, library machines.
- **R4**: Because of the highly experimental nature of this project, over time it may become clear that a chosen approach is not feasible or does not fit the goal. In that case, a different approach may be pursued, as there are many alternative ways to implement the software component, materialised through a diverse array of datasets, public repositories, frameworks and techniques.

Regarding **R1**, the use of virtual environments was enough to eliminate any kind of incompatibility and dependency management became a trivial

task. In addition, even if the project was stored on two different machines and backed up on the cloud, the risks associated with **R2** and **R3** did not occur and the contingency plans were never put in action. On the other hand, issue **R4** regularly arose but, by coding a robust framework for experimentation and adopting a flexible software development methodology, it became more of a necessary part of the research than an inconvenience.

Chapter 6

Conclusions

6.1 Technical Achievement

The overarching goal of proving the feasibility of using continual learning in order to improve the efficiency of Graph Neural Network based recommender systems has been achieved and all the secondary objectives have been attained.

As the outcome of this project is a model that has outperformed all the baselines while at the same time achieving a compelling speed up, this project was undoubtedly successful. Thus it has been shown that a significant amount of data used when retraining is redundant and can be safely discarded when training the model without a big hit on the neural network's performance.

However, it has also been shown that selecting training data in an efficient manner is extremely non-trivial and highly advanced techniques have to be employed. This is because the data selected for training cannot be considered in isolation, but their relationships both in the graph topology space and em-

bedding space have to be considered. As the pure random heuristic has been demonstrated to be much superior to seemingly smarter heuristics, it can be concluded that diversity is a crucial factor in the edge selection process.

6.2 Limitations and Further Work

6.2.1 Hyperparameter Tuning

A clear path of further research is hyperparameter tuning, as the proposed architecture has an overwhelming amount of parameters and the chances of the optimal values being already found are virtually nil. However, this was deemed as extremely labour intensive and would not contribute significantly to the added value of this project.

6.2.2 Timeframe-Related Parameter Optimisation

An important aspect that requires further attention is the robustness of the implemented methods over long periods of time/large amounts of data and finding optimal settings for the implemented solution, i.e. using larger-size timeframes versus smaller-size timeframes or triggering the update when a set amount of new data points are generated versus at fixed time intervals. Finding and quantifying the trade-offs of these different options is an important step towards creating a solution usable in production environments.

6.2.3 Exploring Other Datasets

Nonetheless, the main limitations of this project are strongly tied with the biases of the MovieLens dataset. Because much of the information is concentrated at the start of the dataset and entries from the same users tend to be clustered around the same timeframes, it rewards certain approaches, such as replaying recent data points, while punishing others, such as repeatedly replaying the same data points which are deemed as "important". Hence, a natural next step is applying the developed techniques to different datasets, such as the 25M variant of the MovieLens dataset or the Yelp dataset [47], which contains a subset of Yelp's businesses, reviews, and user data but also to non-recommendation related datasets such as Cora [2], a citation network of 2708 scientific publications classified into seven classes, or the Elliptic dataset [48], which contains transactions on the Bitcoin [49] network and entities labelled as licit or illicit.

6.2.4 Exploring Other Models

At the same time, this project only researched one recommender system architecture, namely the Matapath and Entity Aware Graph Attention Network architecture. There is a need for further investigation into related architectures such as NGCF [50], KGCN [51], KGAT [52], Multi-GCCF [53], PEASage [10], PEAGCN [10], but also non-GNN approaches, such as Neural Collaborative Filtering [54], Neural Factorization Machines for Sparse Predictive Analytics [55] and Collaborative Knowledge Base Embedding for Recommender Systems

[56].

In addition, a further investigation into the improvements gained through the use of metapaths, but also experimentation with entity awareness, suggested in the same paper, could prove useful. Enitity awareness is a contrastive regulariser which pushes vertices linked in the graph closer together in the embedding space, and by strengthening this correlation between the topological and embedded representations of the graph, it might simplify the diversity problem. [10]

6.2.5 Exploring Other Continual Learning Methods

On the flip side, further experimentation with other continual learning techniques is also a subject of interest. Other replay-based methods, e.g. the use of synthetic data points instead of real past data points for the "reminding" process [57], and regularisation-based methods, e.g. "Learning without Forgetting" [58], are available while the class of parameter-isolation methods was completely ignored.

6.2.6 Improving Scalability and Usability

Furthermore, increased speed-ups can be achieved by studying the impact of parallel computing methods, developing techniques for scaling and partitioning the training and inference process for streaming data. This includes leveraging network sparsity by using SpMV-based solutions [59] and partitioning the graph into subgraphs that can be trained and used for inference inde-

pendently and employing message-passing to propagate information between subgraphs [60].

Nevertheless, it can be seen that much emphasis was put on the time complexity aspect when it comes to the performance of the approach, leaving considerable inefficiencies space-complexity wise. Compression methods that can ameliorate this issue are available and can be integrated into the presented solution. One promising approach is using learning vector quantization [61], which is already being studied within our department [62] and has been successfully applied on convolutional neural networks (these being architecturally similar to GNNs), is some cases even increasing the accuracy of the replay method. [63]

Lastly, an alternative route is introducing practicality into the project by integrating the solution with existing industry standard technologies, such as stream processing frameworks (Apache Flink [64]/Kafka [65]) or graph databases (Neo4j [66]) and implementing a production ready pipeline that can be easily integrated into existing workflows.

References

[1] 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*, CIKM '20, page 1515–1524, New York, NY, USA, 2020. Association for Computing Machinery.

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

[3] Yelp Inc. Annual Report. `https://www.annualreports.com/HostedData/AnnualReports/PDF/NYSE_YELP_2020.pdf`. (Accessed on 20/04/2022).

[4] Netflix - Financials - Financial Statements. `https://ir.netflix.net/financials/financial-statements/default.aspx`. (Accessed on 20/04/2022).

[5] Netflix Library A-Z - What's on Netflix. `https://www.whats-on-netflix.com/library/`. (Accessed on 20/04/2022).

[6] YouTube for Press. https://blog.youtube/press/. (Accessed on 20/04/2022).

[7] How google powers its 'monopoly' with enough electricity for entire countries. https://www.forbes.com/sites/robertbryce/2020/10/21/googles-dominance-is-fueled-by-zambia-size-amounts-of-electricity/?sh=4e79509e68c9. (Accessed on 20/04/2022).

[8] Electricity rates by state - energybot. https://www.energybot.com/electricity-rates-by-state.html. (Accessed on 20/04/2022).

[9] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *CoRR*, abs/1810.00826, 2018.

[10] Zhiwei Han, Muhammad Umer Anwaar, Shyam Arumugaswamy, Thomas Weber, Tianming Qiu, Hao Shen, Yuanting Liu, and Martin Kleinsteuber. Metapath- and entity-aware graph neural network for recommendation. *CoRR*, abs/2010.11793, 2020.

[11] Steffen Pauws and Berry Eggen. Realization and user evaluation of an automatic playlist generator. *Journal of New Music Research*, 32(2):179–192, 2003.

[12] Richard Lawrence, George Almasi, Vladimir Kotlyar, M.S. Viveros, and Sastry Duri. Personalization of supermarket product recommendations. *Data Mining and Knowledge Discovery*, 5:11–32, 01 2001.

[13] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on*

Recommender Systems, RecSys '16, page 191–198, New York, NY, USA, 2016. Association for Computing Machinery.

[14] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*, CSCW '94, page 175–186, New York, NY, USA, 1994. Association for Computing Machinery.

[15] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4), December 2015.

[16] James Bennett, Stan Lanning, et al. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35. Citeseer, 2007.

[17] Yizhou Sun and Jiawei Han. Mining heterogeneous information networks: A structural analysis approach. *SIGKDD Explor. Newsl.*, 14(2):20–28, April 2013.

[18] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[19] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. *CoRR*, abs/1706.02216, 2017.

[20] Shiwen Wu, Wentao Zhang, Fei Sun, and Bin Cui. Graph neural networks in recommender systems: A survey. *CoRR*, abs/2011.02260, 2020.

[21] Lingfei Wu, Yu Chen, Kai Shen, Xiaojie Guo, Hanning Gao, Shucheng Li, Jian Pei, and Bo Long. Graph neural networks for natural language processing: A survey. *CoRR*, abs/2106.06090, 2021.

[22] Marinka Zitnik, Monica Agrawal, and Jure Leskovec. Modeling polypharmacy side effects with graph convolutional networks. *Bioinformatics*, 34(13):i457–i466, 06 2018.

[23] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[24] Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, R. Howard, Wayne Hubbard, and Lawrence Jackel. Handwritten digit recognition with a back-propagation network. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann, 1989.

[25] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. In David E. Rumelhart and James L. Mcclelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, pages 318–362. MIT Press, Cambridge, MA, 1986.

[26] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A review of methods and applications. *CoRR*, abs/1812.08434, 2018.

[27] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1263–1272. PMLR, 06–11 Aug 2017.

[28] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S. Yu, and Tianyi Wu. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *Proc. VLDB Endow.*, 4(11):992–1003, August 2011.

[29] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.

[30] S. Ruping. Incremental learning with support vector machines. In *Proceedings 2001 IEEE International Conference on Data Mining*, pages 641–642, 2001.

[31] S. Shalev-Shwartz. *Online Learning: Theory, Algorithms, and Applications*. Hebrew University, 2007.

[32] Ian J. Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks, 2015.

[33] German I. Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, 2019.

[34] James Kirkpatrick, Razvan Pascanu, Neil C. Rabinowitz, Joel Veness,

Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *CoRR*, abs/1612.00796, 2016.

[35] Tyler L. Hayes, Kushal Kafle, Robik Shrestha, Manoj Acharya, and Christopher Kanan. REMIND your neural network to prevent catastrophic forgetting. *CoRR*, abs/1910.02509, 2019.

[36] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. *CoRR*, abs/1705.08690, 2017.

[37] Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks, 2016.

[38] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, and Christoph H. Lampert. icarl: Incremental classifier and representation learning. *CoRR*, abs/1611.07725, 2016.

[39] Razvan Pascanu and Yoshua Bengio. Revisiting natural gradient for deep networks, 2013.

[40] Antonio Carta, Andrea Cossu, Federico Errica, and Davide Bacciu. Catastrophic forgetting in deep graph networks: an introductory benchmark for graph classification. *CoRR*, abs/2103.11750, 2021.

[41] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerat-

ing deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[42] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[43] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[44] Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010.

[45] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernán-

dez del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[46] NVIDIA's Next Generation CUDA Compute Architecture: Fermi. `nvidia.com/content/PDF/fermi_white_papers/NVIDIAFermiComputeArchitectureWhitepaper.pdf`. Accessed: 2021-10-12.

[47] Nabiha Asghar. Yelp dataset challenge: Review rating prediction. *CoRR*, abs/1605.05362, 2016.

[48] Mark Weber, Giacomo Domeniconi, Jie Chen, Daniel Karl I. Weidele, Claudio Bellei, Tom Robinson, and Charles E. Leiserson. Anti-money laundering in bitcoin: Experimenting with graph convolutional networks for financial forensics. *CoRR*, abs/1908.02591, 2019.

[49] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2009.

[50] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. Neural graph collaborative filtering. *CoRR*, abs/1905.08108, 2019.

[51] Hongwei Wang, Miao Zhao, Xing Xie, Wenjie Li, and Minyi Guo. Knowledge graph convolutional networks for recommender systems. *CoRR*, abs/1904.12575, 2019.

[52] Xiang Wang, Xiangnan He, Yixin Cao, Meng Liu, and Tat-Seng Chua. KGAT: knowledge graph attention network for recommendation. *CoRR*, abs/1905.07854, 2019.

[53] Jianing Sun, Yingxue Zhang, Chen Ma, Mark Coates, Huifeng Guo, Ruiming Tang, and Xiuqiang He. Multi-graph convolution collaborative filtering. *CoRR*, abs/2001.00267, 2020.

[54] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. *CoRR*, abs/1708.05031, 2017.

[55] Xiangnan He and Tat-Seng Chua. Neural factorization machines for sparse predictive analytics. *CoRR*, abs/1708.05027, 2017.

[56] Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, and Wei-Ying Ma. Collaborative knowledge base embedding for recommender systems. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 353–362, New York, NY, USA, 2016. Association for Computing Machinery.

[57] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. *CoRR*, abs/1705.08690, 2017.

[58] Zhizhong Li and Derek Hoiem. Learning without forgetting, 2017.

[59] Gunduz Vehbi Demirci and Hakan Ferhatosmanoglu. Partitioning sparse deep neural networks for scalable training and inference. *CoRR*, abs/2104.11805, 2021.

[60] Renjie Liao, Marc Brockschmidt, Daniel Tarlow, Alexander L. Gaunt, Raquel Urtasun, and Richard S. Zemel. Graph partition neural networks for semi-supervised classification. *CoRR*, abs/1803.06272, 2018.

[61] Panu Somervuo and Teuvo Kohonen. Self-organizing maps and learning vector quantization for feature sequences. *Neural Processing Letters*, 10(2):151–159, Oct 1999.

[62] Shuang Wang and Hakan Ferhatosmanoglu. PPQ-Trajectory: Spatio-Temporal Quantization for Querying in Large Trajectory Repositories. *Proc. VLDB Endow.*, 14(2):215–227, oct 2020.

[63] Kai Wang, Luis Herranz, and Joost van de Weijer. ACAE-REMIND for online continual learning with compressed feature replay. *CoRR*, abs/2105.08595, 2021.

[64] Paris Carbone, Asterios Katsifodimos, † Kth, Sics Sweden, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. Apache flink™: Stream and batch processing in a single engine. *IEEE Data Engineering Bulletin*, 38, 01 2015.

[65] Jay Kreps, Linkedin Corp, Neha Narkhede, Jun Rao, and Linkedin Corp. Kafka: a distributed messaging system for log processing. netdb'11.

[66] Jim Webber. A programmatic introduction to neo4j. In *Proceedings of the 3rd Annual Conference on Systems, Programming, and Applications: Software for Humanity*, SPLASH '12, page 217–218, New York, NY, USA, 2012. Association for Computing Machinery.

[67] Yelp Dataset | Kaggle. `kaggle.com/yelp-dataset/yelp-dataset`. Accessed: 2021-10-12.

[68] Xinyu Fu, Jiani Zhang, Ziqiao Meng, and Irwin King. MAGNN: metapath

aggregated graph neural network for heterogeneous graph embedding. *CoRR*, abs/2002.01680, 2020.

[69] Lou Jost. Entropy and diversity. *Oikos*, 113(2):363–375, 2006.

[70] Elif Eser, Tolga Can, and Hakan Ferhatosmanoğlu. Div-BLAST: diversification of sequence search results. *PLoS One*, 9(12):e115445, December 2014.

[71] Kaan Bingöl, Bahaeddin Eravcı, Çağrı Özgenç Etemoğlu, Hakan Ferhatosmanoğlu, and Buğra Gedik. Topic-based influence computation in social networks under resource constraints. *IEEE Transactions on Services Computing*, 12(6):970–986, 2019.

[72] Linyuan Lü, Matúš Medo, Chi Ho Yeung, Yi-Cheng Zhang, Zi-Ke Zhang, and Tao Zhou. Recommender systems. *Physics Reports*, 519(1):1–49, oct 2012.

[73] M. Syeda, Yan-Qing Zhang, and Yi Pan. Parallel granular neural networks for fast credit card fraud detection. In *2002 IEEE World Congress on Computational Intelligence. 2002 IEEE International Conference on Fuzzy Systems. FUZZ-IEEE'02. Proceedings (Cat. No.02CH37291)*, volume 1, pages 572–577 vol.1, 2002.

[74] Alessio Micheli. Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks*, 20(3):498–511, 2009.

[75] Wai Weng Lo, Siamak Layeghy, and Marius Portmann. Inspection-l: A self-supervised gnn-based money laundering detection system for bitcoin, 2022.

[76] Hiroki Kanezashi, Toyotaro Suzumura, Xin Liu, and Takahiro Hirofuchi. Ethereum fraud detection with heterogeneous graph neural networks, 2022.

[77] Benjamin Sanchez-Lengeling, Emily Reif, Adam Pearce, and Alexander B. Wiltschko. A gentle introduction to graph neural networks. *Distill*, 2021. https://distill.pub/2021/gnn-intro.

[78] Sihao Ding, Fuli Feng, Xiangnan He, Yong Liao, Jun Shi, and Yongdong Zhang. Causal incremental graph convolution for recommender system retraining. *CoRR*, abs/2108.06889, 2021.

[79] Xu Chen, Junshan Wang, and Kunqing Xie. Trafficstream: A streaming traffic flow forecasting framework based on graph neural networks and continual learning. *CoRR*, abs/2106.06273, 2021.

[80] Xiaoyu Kou, Yankai Lin, Shaobo Liu, Peng Li, Jie Zhou, and Yan Zhang. Disentangle-based continual graph representation learning. *CoRR*, abs/2010.02565, 2020.

[81] Binh Tang and David S. Matteson. Graph-based continual learning, 2020.

[82] Khurram Javed and Martha White. Meta-learning representations for continual learning. *CoRR*, abs/1905.12588, 2019.

[83] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael M. Bronstein. Temporal graph networks for deep learning on dynamic graphs. *CoRR*, abs/2006.10637, 2020.

[84] Yuxiao Dong, Nitesh V. Chawla, and Ananthram Swami. Metapath2vec: Scalable representation learning for heterogeneous networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, page 135–144, New York, NY, USA, 2017. Association for Computing Machinery.

[85] Avishek Joey Bose, Ankit Jain, Piero Molino, and William L. Hamilton. Meta-graph: Few shot link prediction via meta learning. *CoRR*, abs/1912.09867, 2019.

[86] Chuan Shi, Binbin Hu, Wayne Xin Zhao, and Philip S. Yu. Heterogeneous information network embedding for recommendation. *CoRR*, abs/1711.10730, 2017.

[87] Yifan Wang, Suyao Tang, Yuntong Lei, Weiping Song, Sheng Wang, and Ming Zhang. Disenhan: Disentangled heterogeneous graph attention network for recommendation. *CoRR*, abs/2106.10879, 2021.

[88] Xiaohan Li, Mengqi Zhang, Shu Wu, Zheng Liu, Liang Wang, and Philip S. Yu. Dynamic graph collaborative filtering. *CoRR*, abs/2101.02844, 2021.

[89] Lukas Galke, Iacopo Vagliano, and Ansgar Scherp. Incremental training of graph neural networks on temporal graphs under distribution shift. *CoRR*, abs/2006.14422, 2020.

[90] Yishi Xu, Yingxue Zhang, Wei Guo, Huifeng Guo, Ruiming Tang, and Mark Coates. Graphsail: Graph structure aware incremental learning for recommender systems. *CoRR*, abs/2008.13517, 2020.

[91] Martin Junghanns, André Petermann, Niklas Teichmann, Kevin Gómez, and Erhard Rahm. Analyzing extended property graphs with apache flink. In *Proceedings of the 1st ACM SIGMOD Workshop on Network Data Analytics*, NDA '16, New York, NY, USA, 2016. Association for Computing Machinery.

[92] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Peng Cui, Philip S. Yu, and Yanfang Ye. Heterogeneous graph attention network. *CoRR*, abs/1903.07293, 2019.

[93] Palash Goyal, Nitin Kamra, Xinran He, and Yan Liu. Dyngem: Deep embedding method for dynamic graphs. *CoRR*, abs/1805.11273, 2018.

[94] Yao Ma, Ziyi Guo, Zhaocun Ren, Jiliang Tang, and Dawei Yin. *Streaming Graph Neural Networks*, page 719–728. Association for Computing Machinery, New York, NY, USA, 2020.

[95] Khurram Javed and Martha White. Meta-learning representations for continual learning. *CoRR*, abs/1905.12588, 2019.

[96] Paul Resnick and Hal R Varian. Recommender systems. *Communications of the ACM*, 40(3):56–58, 1997.

Appendix A

Accepted Command Line Arguments

Parameter	Default value	Explanation
dataset	MovieLens	the dataset
dataset_name	latest-small	the dataset variant
if_use_features	false	use features as default embeding
num_core	10	users must have at least this no of rating
num_feat_core	10	entity has to occur at least this no of times
sampling_strategy	unseen	negative examples sampling strategy: unseen/random
entity_aware	false	employ entity awareness
dropout	0	dropout probability
emb_dim	64	embedding dimensionality
num_heads	1	number of heads in multi-head attention
repr_dim	16	representation dimensionality
hidden_size	64	hidden layer width
meta_path_steps	2,2,2,2,2,2,2,2,2	no of steps for each metapath
channel_aggr	att	channel aggregation method: concat/mean/att
entity_aware_coff	0.1	entity awareness coefficient
init_eval	true	perform evaluation at initialisation
num_negative_samples	4	no of negative samples used for training for each positive example
num_neg_candidates	99	number of negative examples used as candidates to be ranked during testing
device	cuda	pytorch device
gpu_idx	0	pytorch gpu id
runs	1	no of repeats
epochs	30	no of epochs for training
batch_size	1024	batch size
num_workers	12	pytorch DataLoader no of workers
opt	adam	optimiser: adam/sgd/sparseadam
lr	0.001	learning rate
weight_decay	0.001	weight decay rate
save_epochs	5,10,15,20,25	save parameters in these epoch
save_every_epoch	26	save parameters every this no of epochs
metapath_test	false	perform metapath contribution test
continual_aspect	continual	specify base algorithm: retrained/ pretrained/single/online/continual
equal_timespan_timeframes	false	use equal times span timeframes
num_timeframes	25	split into this no of timeframes
end_timeframe	None	end at this timeframe (None means end at last timeframe)
start_timeframe	0	start with this timeframe
ewc_type	ewc	regularisation type: ewc/l2
ewc_lambda	80	ewc coefficient
theta	1.5	replay-method theta parameter
future_testing	true	perform future edge detection
train_between_emb_diff	false	add a training step in between inference steps when calculating importance
out_filename	out_filename	concatenate this at the end of the generated file with results (default: empty)

Appendix B

Plots with the Performance of the Tested Heuristics

The plots for the heuristics in the no better than random group were skipped as these are identical to the plot of the *random* heuristic.

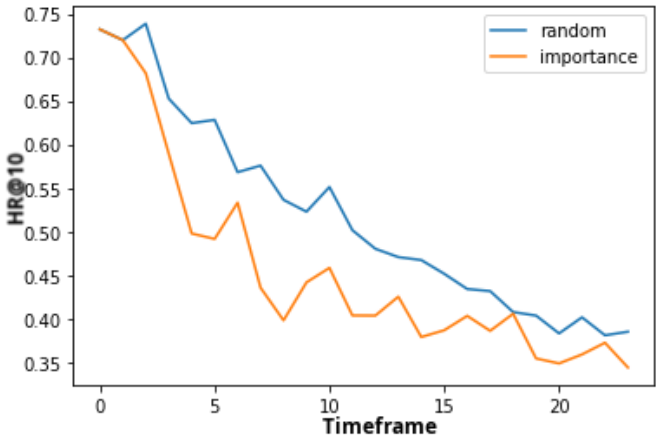


Figure B.1: The evolution of the model's hit ratio over time when using the *importance* heuristic

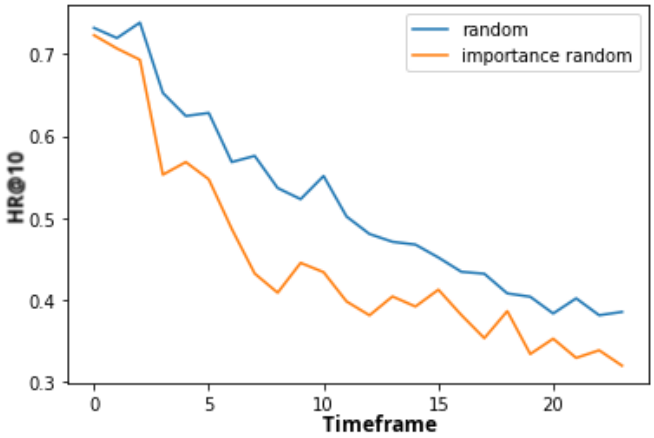


Figure B.2: The evolution of the model's hit ratio over time when using the *random by importance* heuristic

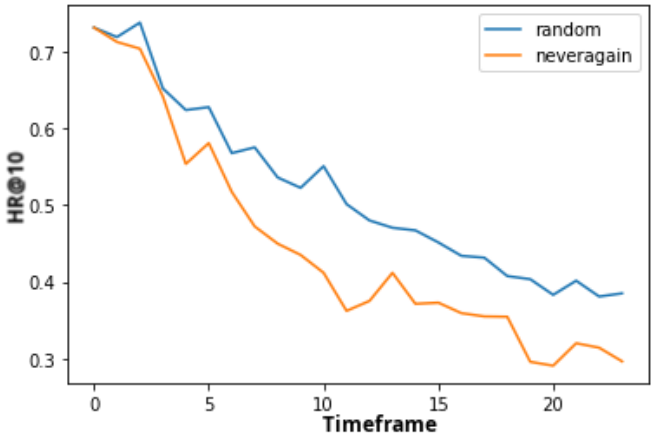


Figure B.3: The evolution of the model's hit ratio over time when using the *never again* heuristic

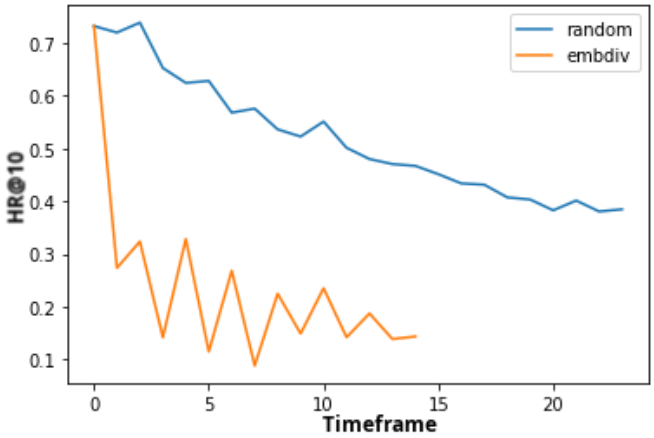


Figure B.4: The evolution of the model's hit ratio over time when using the *embedding diversity* approach (program was interrupted after timeframe 15 as the performance was already clearly inferior)

Appendix C

List of Used Metapaths

- User-Movie-User
- Movie-User-Movie
- Year-Movie-User
- Actor-Movie-User
- Writer-Movie-User
- Director-Movie-User
- Genre-Movie-User
- Tag-Movie-User
- Tag-User-Movie

Appendix D

Project Specification

CS310 Project Specification: Continual Learning Using Metapath Aware Graph Neural Networks

Ion Orins
1925873

1 Problem Statement

The objective of this project is to develop a graph neural network (GNN) solution that can provide continual learning support. Currently, GNNs are proven to be exceedingly useful and versatile in big data applications. Recently metapath aware GNNs have been shown to be one of the best performing architectures of such neural networks. At the same time batch processing techniques start to show their flaws as huge amounts of data have to be processed every second, continual learning rising in its popularity. However, metapath aware GNNs have never been used in the context of continual learning, which is the goal this project is set to achieve.

2 Literature Review

An exceeding number of phenomena can be modelled as networks and, undoubtedly, there is a tendency to use homogeneous graphs as the modelling tool. However, relationships between entities are often complex and a better approach is often the use of heterogeneous networks[1].

A nice application compatible with this concept are GNNs[2]. For example, message passing GNN architectures, such as GraphSAGE[3], naturally incorporate the heterogeneity of the graph by taking into account the type of connection when passing information between two nodes.

Moreover, it has been shown that an effective method to improve GNNs' performance is to augment them with metapath awareness, achieving state of art performance on recommender systems[4]. Metapaths are "sequences of relations defined between different object types"[5], and by considering them the neural network can be aware of the deeper semantics of the relationship between two nodes further apart in the network.

In parallel, researchers have been trying to overcome the issue of "catastrophic forgetting"[6] in neural networks, by developing a number of techniques, commonly referred to as "continual learning":

- Elastic weight consolidation[7], which tries to find the middle ground between the optimum of the previous and the current tasks;

- Progressive neural networks[8], which adds new neurons for the new task and does not modify the old weights;
- Generative replay[9], which generates synthetic data that resembles the distribution of the old data and uses it alongside the new data.

Some of these have been successfully applied to GNNs, such as ContinualGNN[10], which uses elastic weight consolidation.

3 Objectives

1. Adapt a graph neural network architecture compatible with continual learning to a chosen dataset for recommender systems.
2. Evaluate the performance of said GNN.
3. Adapt a metapath aware graph neural network to the same dataset for recommender systems.
4. Evaluate the performance of said metapath aware GNN.
5. Implement a metapath aware graph neural network architecture that is compatible with continual learning.
6. Evaluate its performance on the same dataset as the previous two architectures.

4 Possible extensions

- Integrate with stream processing software in order to create a production-ready pipeline.
- Improve the accuracy through hyperparameter tuning.
- Improve accuracy using different graph neural network techniques, such as entity awareness[4].
- Improve performance using parallel computing methods, such as CUDA[11].
- Explore other applications of continual learning GNNs (e.g.: fraud detection, citation network classification).
- Experiment with other continual learning techniques (e.g.: learning without forgetting, elastic weight consolidation, progressive neural networks, generative replay).

5 Methodology

As this project falls under the research category but also has a software development component, its complexity prohibits a plan-driven approach, Agile's increased flexibility being optimal. As there are many techniques to test, plenty of potential points of failure, and a considerable number of possible routes to be explored, the author considered that the Kanban methodology will fit best the roadmap of the project. This allows for quickly updating the backlog of tasks (adding new ideas and removing tasks that became obsolete), pausing and switching between diverging approaches, as their versatility is being discovered and deadends are reached.

A useful tool for approach will be git, as its tree structure and merging functionality reflect the philosophy of Kanban. In addition, by using a hosting provider, such as GitHub, it will double as a backup service of the source code.

The testing component of the project will be done using publicly available datasets, following the well known train - test - validation split methodology.

6 Risk Assessment

The following risks have been identified, with their corresponding contingency / mitigation plans:

- As frameworks and public repositories with a high number of dependencies will be used, there is a great chance that software incompatibilities may arise. In order to mitigate this, isolation techniques such as virtual environments, containers and virtual machines can be used.
- As the risk of data loss is present, just like in any software project, backups will be used and data will be stored both locally and in the cloud.
- A hardware failure that restricts access to development may occur. Fortunately, as data is backed up, there is always an alternative to resume development: personal laptop, DCS machines, library machines.
- Because of the highly experimental nature of this project, over time it may become clear that a chosen approach is not feasible or does not fit the goal. In that case, a different approach may be pursued, as there are many alternative ways to implement the software component, materialised through a diverse array of datasets, public repositories, frameworks and techniques.

7 Timeline

Term 1

Week 1	agree on topic, initiate project
Week 2	**project specification**, successfully run a continual learning GNN
Week 3	adapt a dataset to a continual learning GNN
Week 4	successfully run a metapath aware GNN
Week 5	implement a metapath aware continual learning GNN
Week 6	evaluate implemented GNN
Weeks 7/8	organise and better structure project, improve code readability
Week 9	**progress report**
Week 10	explore paradigms to upgrade the proof of concept to a production-ready pipeline

Term 2

Weeks 1-4	implement production-ready pipeline
Weeks 5/6	explore feasible project extensions
Weeks 7/8	implement extensions
Weeks 9/10	**presentation**

Term 3

Week 1	**final report**

8 Resources

8.1 Hardware

Hardware-wise, a personal laptop will be used to write the documentation, implement short codebases intended as proof of concepts and also to interface with DCS machines, on which the main bulk of work will be done.

8.2 Software

The main programming language to be used will be Python, with the help of various machine learning and data science libraries (e.g.: PyTorch[12], scikit-learn[13], pandas[14], NumPy[15]). Occasionally, short automation scripts will be written using a shell language, such as Bash.

8.3 Public Code

Multiple public repositories with open-source implementations of concepts that are going to be used in this project are available online, which can help accelerate

the code writing process for the proof of concept. More specifically, the following two repositories seem the most promising:

- ContinualGNN[10] - which offers an implementation of elastic weight consolidation on message passing graph neural networks;
- PEAGNN[4] - which offers implementations of multiple architectures of metapath- and entity-aware graph neural networks.

8.4 Datasets

This project mainly focuses on recommender systems in contexts that can be modelled as graphs. The following freely available datasets that fit the requirements have been identified:

- MovieLens[16] - a dataset that contains film ratings, as well as attributes about both users and the rated films;
- Yelp Dataset[17] - "a subset of Yelp's businesses, reviews, and user data"[18].

9 Ethical Considerations

There are no ethical concerns to be considered.

References

[1] Yizhou Sun and Jiawei Han. Mining heterogeneous information networks: A structural analysis approach. *SIGKDD Explor. Newsl.*, 14(2):20–28, April 2013.

[2] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[3] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. *CoRR*, abs/1706.02216, 2017.

[4] Zhiwei Han, Muhammad Umer Anwaar, Shyam Arumugaswamy, Thomas Weber, Tianming Qiu, Hao Shen, Yuanting Liu, and Martin Kleinsteuber. Metapath- and entity-aware graph neural network for recommendation. *CoRR*, abs/2010.11793, 2020.

[5] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S. Yu, and Tianyi Wu. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *Proc. VLDB Endow.*, 4(11):992–1003, August 2011.

[6] Ian J. Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks, 2015.

[7] James Kirkpatrick, Razvan Pascanu, Neil C. Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *CoRR*, abs/1612.00796, 2016.

[8] Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks, 2016.

[9] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. *CoRR*, abs/1705.08690, 2017.

[10] 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*, CIKM '20, page 1515–1524, New York, NY, USA, 2020. Association for Computing Machinery.

[11] NVIDIA's Next Generation CUDA Compute Architecture: Fermi. `nvidia.com/content/PDF/fermi_white_papers/NVIDIAFermiComputeArchitectureWhitepaper.pdf`. Accessed: 2021-10-12.

[12] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[13] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[14] Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010.

[15] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane,

Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[16] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4), December 2015.

[17] Nabiha Asghar. Yelp dataset challenge: Review rating prediction. *CoRR*, abs/1605.05362, 2016.

[18] Yelp Dataset — Kaggle. `kaggle.com/yelp-dataset/yelp-dataset`. Accessed: 2021-10-12.

Appendix E

Progress Report

CS310 Progress Report: Continual Learning Using Metapath Aware Graph Neural Networks

Ion Orins
1925873

1 Introduction

The aim of this project is to bring improvements to recommender systems based on graph neural networks[1]. More specifically, the goal is to add a continual learning aspect to a metapath[2] aware graph neural network achitecture, which are regarded as being the state of art architecture for recommender systems[3].

2 Background

The project is mainly based on the "Metapath- and Entity-aware Graph Neural Network for Recommendation" (PEAGNN) paper[3], which provides the theoretical framework for the neural network architecture to be improved and an example implementation which was used as the basis for this project.

The chosen continual learning approach, which will be added to the previously mentioned GNN architecture, is called "elastic weight consolidation" (EWC) and was first described in the paper "Overcoming catastrophic forgetting in neural networks". It is designed to enable online learning "by selectively slowing down learning on the weights important for"[4] the old tasks.

In addition, the paper "Streaming Graph Neural Networks via Continual Learning" provides a useful reference on how to successfully implement "elastic weight consolidation" in the context of graph neural networks (again, both as theoretical framework and actual example implementation).

The MovieLens dataset[5] was chosen to be used as the benchmark for the developed algorithms, because it offers a range of different dataset sizes and incorporates nicely the graph heterogeneity aspect, required for the metapath approach to work. It mainly consists out of users, movies (representing the vertices) and reviews given by users to movies (representing the edges).

3 Progress

Out of all the reference implementations provided by the PEAGNN repository, the one using graph attention networks (GAT)[6] has been chosen to be worked

on, because implementations both with and without metapath awareness are provided. Based on this, a working prototype has already been implemented.

Firstly, the loss function of the architecture was updated in accordance with the EWC method, i.e. big changes in the parameters of the neural network are being penalised.

Next, the MovieLens dataset was split into timeframes based on the timestamps of the reviews (for example, a timeframe containing the reviews written in January, one for February and one for March).

Lastly, the training routine was updated to train on each of the timeframes sequentially, but only do so on the vertices adjacent to the edges added to the network in the current timeframe, thus reducing the workload and fulfilling the continual learning requirement.

These changes alone did result in a slight increase in accuracy, however we chose not to provide the actual values, as we have not validated yet that our testing methodology is indeed sound. A pseudocode summary of the implementation is provided in figure 1.

```
for timeframe in timeframes:
    model.add_edges(timeframe.reviews)
    model.train(timeframe.users + timeframe.movies)
    # where the used loss function is
    # model.loss_function_for_current_task(model.current_parameters) +
    # model.ewc_function(model.current_parameters - model.saved_parameters)
    model.save_parameters()
```

Figure 1: A Pseudocode Summary of the Implementation of the Continual Learning Algorithm

Regarding the timeline devised in the Project Specification document, as the goal of implementing a metapath aware continual learning GNN has been achieved and right now the author is working on improving the code, progress is on track.

4 Next Steps

As stated in the previous section, validating the testing methodology is the greatest priority at the moment. In addition, the short term objectives are as follows:

- the paper "Streaming Graph Neural Networks via Continual Learning" mentions the concept of "influenced nodes", vertices from the graph which have not been updated in the current timeframe, but still deemed as important and added to the training set; we are working adapting this concept to work with metapath aware GNNs.

- measure the impact of the implemented changes on the time required to train the network;
- write documentation for the code.

Long term, we considered that it is more important to focus on the research aspect of the project, in contrast with the focus on applications which is present in the Project Specification Document. A key concept that will be explored, once a satisfactory prototype is completed, are performance comparisons with other approaches, such as naïve methods (no EWC or continual learning in general), removing metapath awareness, adding entity awereness, having smaller/bigger timeframes, etc., in order to prove the added value of this project.

5 Thoughts on the Project Management Aspect

The Kanban approach was optimal, just like it was discussed in the Project Specification document, as the priorities of the project change quickly depending on the obtained results and the experimental and research nature of the project make it difficult to create a plan that can be realistically followed. Nonetheless, the following reference timeline was created:

- Term 1 Week 9 - Term 2 Week 2: improve on the prototype;
- Term 2 Week 3 - Term 2 Week 8: implement and evaluate alternative architectures/methodologies;
- Term 2 Week 9 - Term 2 Week 10: presentation;
- Term 3 Week 1: final report;

It is also important to mention that a GitHub repository is being used for version control and weekly meetings were conducted with PhD students working on related projects under the supervision of Prof. Hakan Ferhatosmanoglu in order to gather feedback on progress, further actions and better familiarise with the technical concepts applied in this project.

References

[1] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[2] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S. Yu, and Tianyi Wu. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *Proc. VLDB Endow.*, 4(11):992–1003, August 2011.

[3] Zhiwei Han, Muhammad Umer Anwaar, Shyam Arumugaswamy, Thomas Weber, Tianming Qiu, Hao Shen, Yuanting Liu, and Martin Kleinsteuber. Metapath- and entity-aware graph neural network for recommendation. *CoRR*, abs/2010.11793, 2020.

[4] James Kirkpatrick, Razvan Pascanu, Neil C. Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *CoRR*, abs/1612.00796, 2016.

[5] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4), December 2015.

[6] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.

Appendix F

Final Presentation

Dynamic Learning for Recommender Systems

Recommender Systems

20 million reviews submitted in 2020

222 million subscribers and 15,000 titles

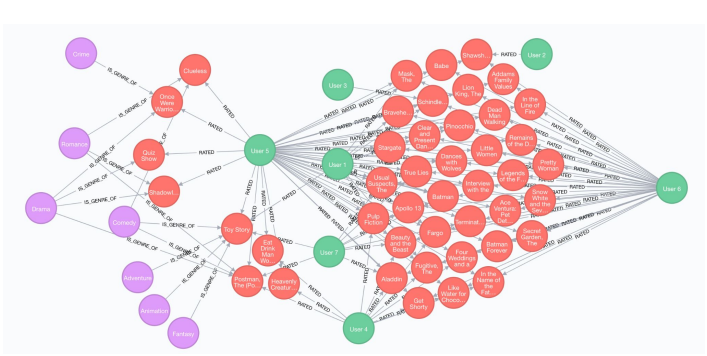
2.3 billion monthly active users

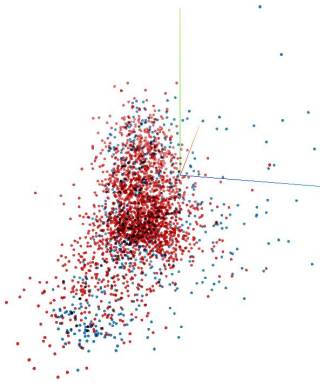
Goal: prevent Catastrophic Forgetting in High Data Throughput scenarios

Overview

Use graph neural networks to perform future edge prediction in a streaming setting using incremental learning

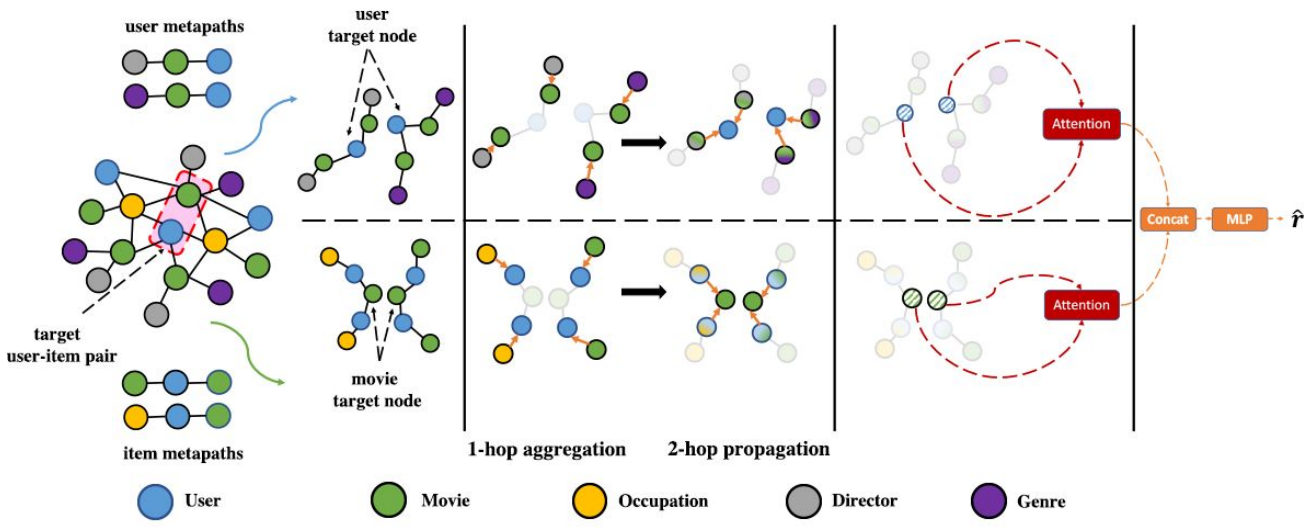
Graph Neural Networks





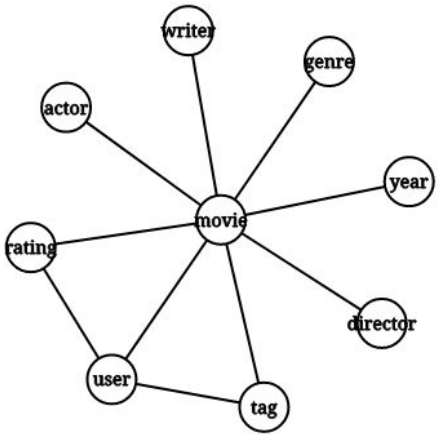
Edge Likelihood Prediction

PEAGAT



MovieLens Dataset

MovieLens-small dataset
Consists of 100,000 ratings

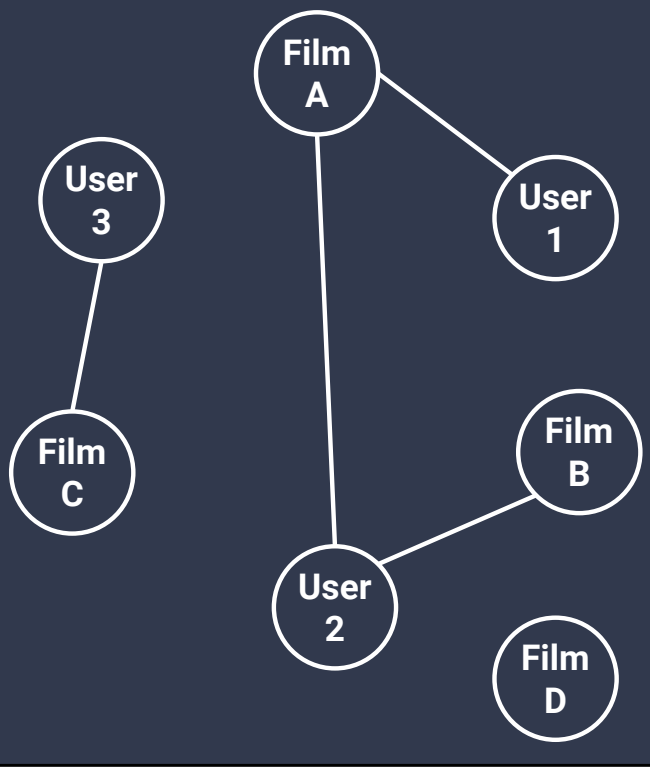


Timeframes

Timeframe 1
Jan 11th - **User 1** rates **Movie A** 5*
Jan 15th - **User 2** rates **Movie B** 3*
Jan 27th - **User 3** rates **Movie A** 4*

Timeframe 2
Feb 11th - **User 1** rates **Movie B** 2*
Feb 15th - **User 4** rates **Movie C** 1*
Feb 27th - **User 4** rates **Movie B** 5*

Timeframe 3
Mar 11th - **User 3** rates **Movie B** 1*
Mar 15th - **User 2** rates **Movie D** 2*
Mar 27th - **User 4** rates **Movie D** 3*



Timeframe 1

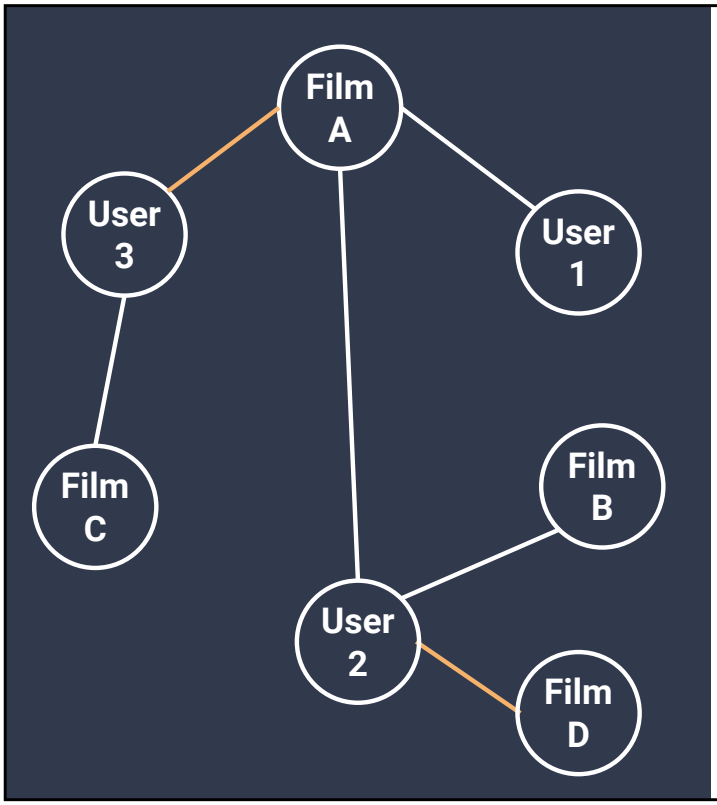
User 1 rated Film A

User 2 rated Film A

Timeframe 2

User 2 rated Film B

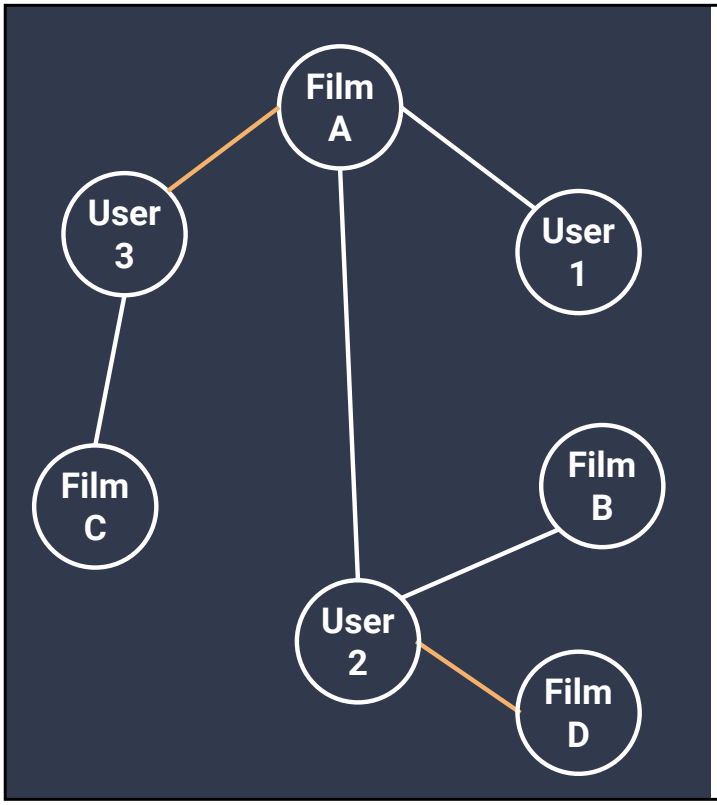
User 3 rated Film C



Timeframe 3

User 2 rated Film D

User 3 rated Film A



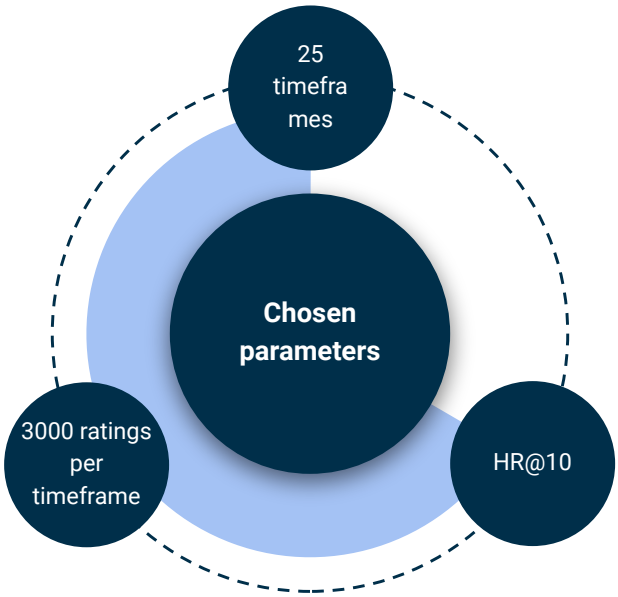
Hit Ratio at 2

Correct guesses / Total guesses

= 1 / 2

= .5

User 2	User 3
1. Film C	1. Film B
2. Film D	2. Film D
3. Film B	3. Film A



Incremental Learning

When input data is continuously used to extend the existing model's knowledge

Continual Learning

When a model learns an increasing amount of tasks without a significant hit on the performance of previously learned tasks

Regularisation

Add terms to the loss function that attempts to penalise the neural network when parameter updates decrease the performance on previous tasks.

Replay

Use past or synthetic data points to "remind" the network about previous tasks.

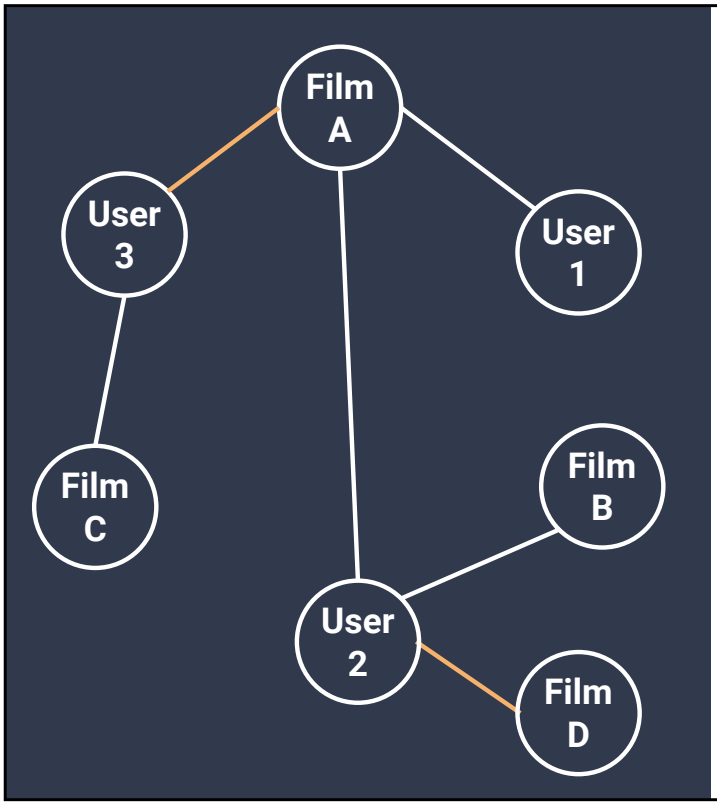
Parameter Isolation

Different parts of a neural network can learn distinct tasks. One proposed approach is starting with a simpler architecture and progressively adding new features such as neurons or layers, whose purpose is to learn the new task.

Continual Approach

Timeframe n: new edges (E_{new})

1. Create E_{old} of size $(\theta\text{-}1)*|E_{new}|$
2. Finetune the model from timeframe n-1 on $E_{old} \cup E_{new}$



Timeframe 3

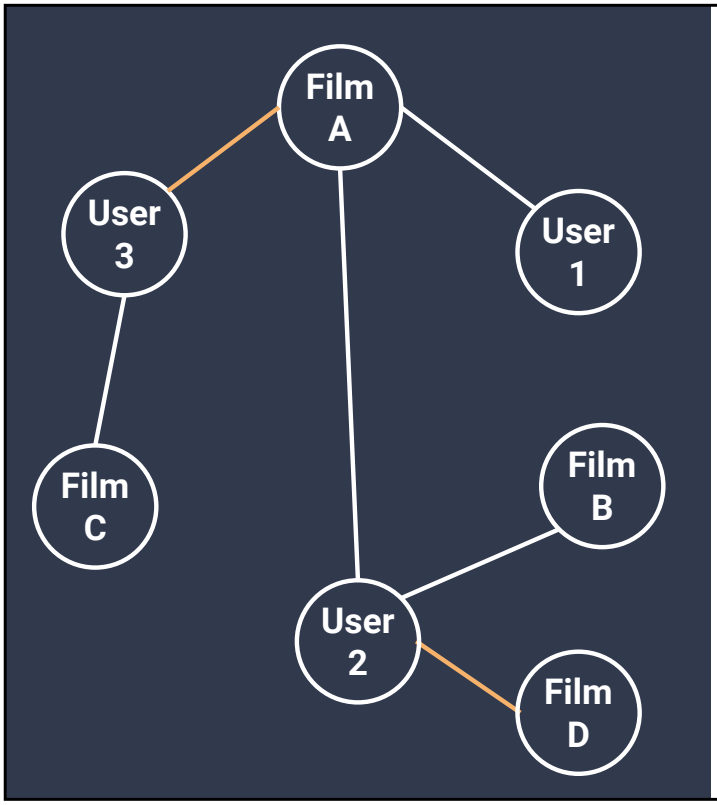
User 2 watched Film D

User 3 watched Film A

Hyperparameter θ

#edges used for training =

#new edges * θ



Timeframe 3

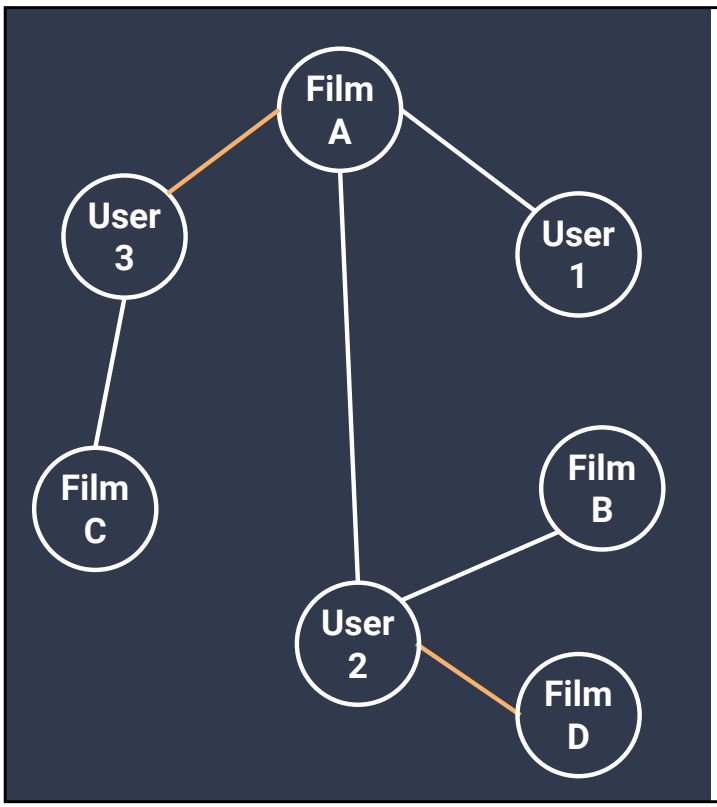
User 2 watched Film D

User 3 watched Film A

θ = 1.5

#edges used for training =

2 * 1.5 = 3 = 2 + 1



Timeframe 3

User 2 watched Film D

User 3 watched Film A

$\theta = 1.5$

#edges used for training =

$2 * 1.5 = 3 = 2 + 1$

Chosen based on heuristic

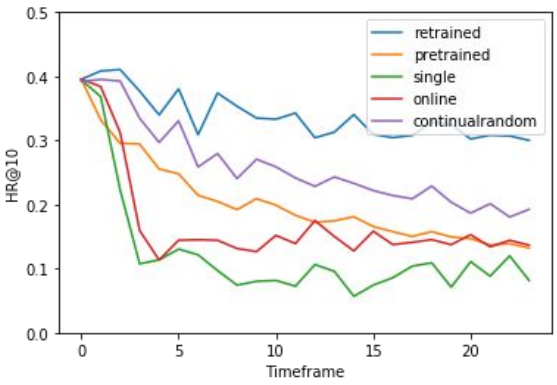
Importance

1. Run inference => $[\text{ ----- e ----- }]_{pre}$
2. Update the topology of the graph
3. Run inference => $[\text{ ----- e ----- }]_{post}$
4. $[\text{ ----- e ----- }]_{post} - [\text{ ----- e ----- }]_{pre} = \Delta[\text{ ----- e ----- }]$
5. $||\Delta[\text{ ----- e ----- }]||$

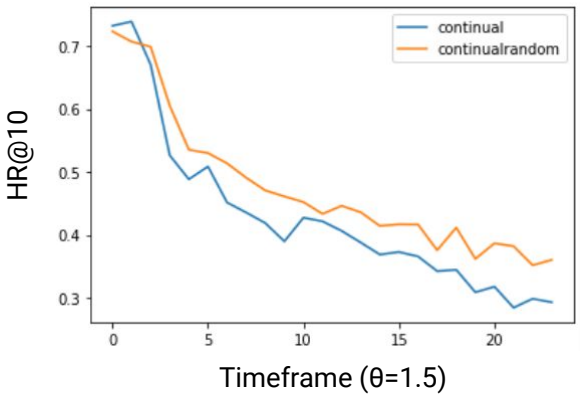
Baselines

Retrained	Use all available data
Pretrained	Train on first timeframe
Single	Train on current timeframe
Online	Continue training on current timeframe

Results (θ=1.5)



Importance



Other Heuristics

Importance: use top k edges based on the value of ||Δh(e)||; results significantly worse than random

Sliding window: only use edges from the last k timeframes randomly (tried with k=5 and k=2); results significantly worse than random

Never use again: once an edge is reused for training, it is marked and never used again; results significantly worse than random

Random by Importance: select edges randomly proportional to their importance; results significantly worse than random

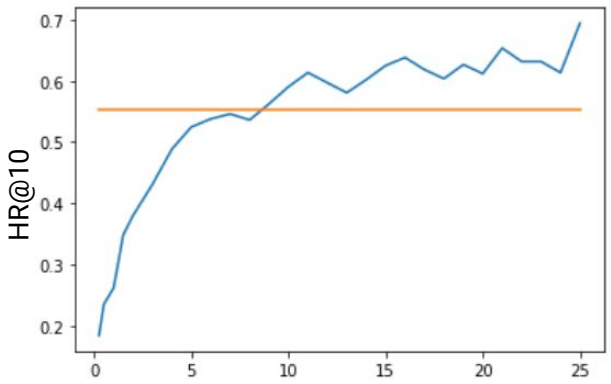
Random by Age: select edges randomly proportional to $2^{age(e)}$; results slightly better than random

Mixed Random: mixes the two approaches above; at best performs slightly better than random (when the proportion of random by age approaches 1 and random by age approaches 0)

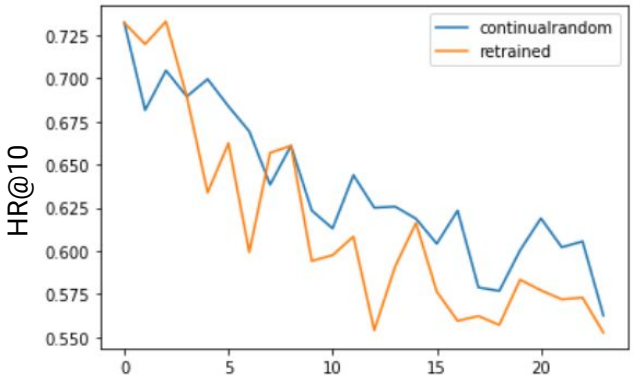
Softmax: chooses edges proportional to the formula below, where b, epsilon and tau are hyperparameters; at best performs just as well as random

$$softmax(\frac{||\Delta h(e)|| * b^{age(e)}}{\tau} + \varepsilon)$$

Adjusting Theta

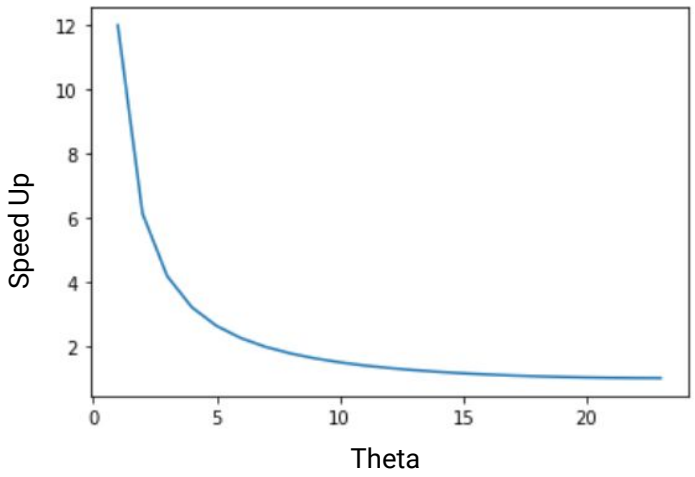


Theta
(timeframe=25)



Timeframe
(θ=9)

Speed Up w.r.t. Retrained



Theta

$$\frac{\sum_{i=0}^{tf} i}{\sum_{i=0}^{tf} min(i, \theta)}$$

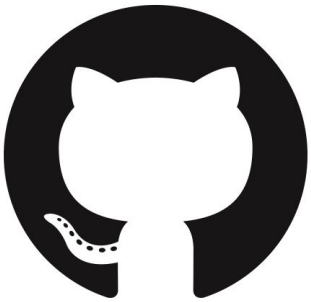
tf - #timeframes

For θ=9

Retrained time: 131 min
Continual time: 95 min

Expected speed up: 1.61
Actual speed up: 1.38

Project Management



Weekly meetings

Timeline

Term 1

Develop robust framework

Term 2

Experiment

Framework

Elastic weight consolidation

Different timeframe sizes

Skipping timeframes

Equal time span timeframes

Adding a training step in between when calculating importance

Different definitions for importance

Batch normalisation

Principal component analysis on node embeddings

Ability to run multiple instances of the algorithm in parallel

One hot encoding for node embeddings

Disabling metapaths

Entity awareness

Selected edge distribution

Different metrics

Discussions and Conclusion

- Outperformed the baseline
- Much of the data is redundant
- Diversity is important
- Experiment on different datasets
- Experiment with other methods
- Space complexity

Special Thanks

Aparajita Haldar

Rustam Guliyev

Thank you

References

- "Graph Neural Networks" slide - Anwaar, Muhammad & Han, Zhiwei & Arumugaswamy, Shyam & Khan, Rayyan & Weber, Thomas & Qiu, Tianming & Shen, Hao & Liu, Yuanting & Kleinsteuber, Martin. (2021). Metapath and Entity-Aware Graph Neural Network for Recommendation.

- "PEAGAT" slide - David Takacs. (2018). MovieLens in Neo4j. Retrieved from https://github.com/tkcsdvd/neo4j-movielens