Degree Project in Computer Science and Engineering

Second cycle, 30 credits

Lightweight Memory Networks for Link Prediction

Generalization of the LiMNet Architecture

TITOUAN MAZIER

Lightweight Memory Networks for Link Prediction

Generalization of the LiMNet Architecture

TITOUAN MAZIER

Master's Program, Computer Science
Date: May 30, 2025

Supervisors: Šarūnas Girdzijauskas and Lodovico Giaretta
Examiner: Viktoria Fodor
School of Electrical Engineering and Computer Science
Host company: RISE, Research Institute of Sweden
Swedish title: Lightweight Memory Networks för Länkprediktion
Swedish subtitle: Generalization av LiMNet Arkitektur

Abstract

- User-item recommendation is a central problem for search engines, social medias and streaming services. Yet, it is challenging because it necessitate to deal with information that is both relational and evolving over time. And common solutions have commonly been neglecting one of these aspects.
- In this work, we try to address these limitations by using the recent Lightweight Memory Networks (LiMNet). A model that captures causal relationships within temporal interaction.
- We demonstrate the potential of this solution throughout a framework for user-item recommendation that allow us to compare the performance of LiMNet with other baselines. The datasets used for the experiments record edits on Wikipedia pages, edits on Reddit pages and music streams on LastFM website. Each of these dataset presenting different scales and challenges for user-item recommendation.

An abstract is (typically) about 250 and 350 words (1/2 A4-page) with the following components:

- What is the topic area? (optional) Introduces the subject area for the project.
- Short problem statement
- Why was this problem worth a Bachelor's/Master's thesis project? (i.e., why is the problem both significant and of a suitable degree of difficulty for a Bachelor's/Master's thesis project? Why has no one else solved it yet?)
- How did you solve the problem? What was your method/insight?
- Results/Conclusions/Consequences/Impact: What are your key results/ conclusions? What will others do based on your results? What can be done now that you have finished - that could not be done before your thesis project was completed?

Keywords

Graph Representation Learning, Temporal Interaction Networks, Link Prediction, Data Mining, Machine Learning, Recommendations

Sammanfattning

Inside the following scontents environment, you cannot use a includefilename as the command rather than the file contents will end up in the for DiVA information. Additionally, you should not use a straight double quote character in the abstracts or keywords, use two single quote characters instead.

Alla avhandlingar vid KTH måste ha ett abstrakt på både engelska och svenska. Om du skriver din avhandling på svenska ska detta göras först (och placera det som det första abstraktet) - och du bör revidera det vid behov.

If you are writing your thesis in English, you can leave this until the draft version that goes to your opponent for the written opposition. In this way, you can provide the English and Swedish abstract/summary information that can be used in the announcement for your oral presentation. If you are writing your thesis in English, then this section can be a summary targeted at a more general reader. However, if you are writing your thesis in Swedish, then the reverse is true – your abstract should be for your target audience, while an English summary can be written targeted at a more general audience. This means that the English abstract and Swedish sammnfattning or Swedish abstract and English summary need not be literal translations of each other.

Do not use the glspl{} command in an abstract that is not in English, as my programs do not know how to generate plurals in other languages. Instead, you will need to spell these terms out or give the proper plural form. In fact, it is a good idea not to use the glossary commands at all in an abstract/summary in a language other than the language used in the acronyms.tex file - since the glossary package does not support use of more than one language.

The abstract in the language used for the thesis should be the first abstract, while the Summary/Sammanfattning in the other language can follow

Nyckelord

Graph Representation Learning, Temporal Interaction Networks, Link Prediction, Data Mining, Machine Learning, Recommendations

Acknowledgments

It is nice to acknowledge the people that have helped you. It is also necessary to acknowledge any special permissions that you have gotten – for example, getting permission from the copyright owner to reproducea figure. In this case, you should acknowledge them and this permission here and in the figure's caption. Note: If you do not have the copyright owner's permission, then you cannot use any copyrighted figures/tables/…. Unless stated otherwise all figures/tables/…are generally copyrighted.

I detta kapitel kan du ev nämna något om din bakgrund om det påverkar rapporten på något sätt. Har du t ex inte möjlighet att skriva perfekt svenska för att du är nyanländ till landet kan det vara på sin plats att nämna detta här. OBS, detta får dock inte vara en ursäkt för att lämna in en rapport med undermåligt språk, undermålig grammatik och stavning (t ex får fel som en automatisk stavningskontroll och grammatikkontroll kan upptäcka inte förekomma) En dualism som måste hanteras i hela rapporten och projektet

Stockholm, May 2025
Titouan Mazier

Contents

List of Figures

List of Tables

Acronyms and Abbreviations

CTDG – **Continuous-Time Dynamic Graph**: Dynamic graph were each edge is associated with a timestamp. 7

GRL – **Graph Representation Learning**: The task of computing high level representation of graphs, subgraphs, nodes or edges. Used as inputs for Machine Learning algorithms. 6, 7

GRU – **Gated Recurrent Unit**: A type of recurrent neural network that simplifies the LSTM architecture by combining the forget and input gates into a single update gate. 9

LiMNet – **Lightweight Memory Networks**: A neural network architecture that focuses on efficient memory utilization and lightweight design for temporal interaction network embeddings computations. vii, ix, 2, 3, 9, 10, 11, 13, 16, 17, 18, 21, 22, 23, 24, 27, 28

LSTM – **Long-Short Term Memory**: A type of recurrent neural network designed to effectively learn and remember short and long-term dependencies in sequential data by using specialized memory cells and gating mechanisms. 9

MRR – **Mean Reciprocal Rank**: Metric used to compute the success of a ranking algorithm. It is a value comprised between 0 and 1 where 0 indicates a total absence of the expected result in the ranked items and 1 mean that the expected item is always ranked first. 22, 25

RNN – **Recurrent Neural Network**: A neural network whose output depends on both the inputs and on the state of an internal memory that is updated with each input. 8, 9

Chapter 1
Introduction

1.1. Purpose/Motivation

The rapid expansion of digital technologies has produced an overwhelming abundance of information, making it challenging to find relevant and meaningful material among the multitude. Thus, search engines and recommendation systems have become essential tools to alleviate this information overload and leverage the large pools of data. These two tools share one common goal: filtering information. Among the many techniques that have emerged to tackle this task, content personalization has emerged as a significant factor. Instead of filtering the information in the same way for everyone, the systems will use the user's context, e.g. search history, demographics, and past interactions with the system, to filter the information to display. Content personalization is the core of recommendation systems, but it is also very efficient for search engines. For example, the search for the term "football" should yield different results for a user interested in American football and a user interested in association football (soccer).

Content personalization can be represented as a single algorithm that accepts as input user-related information and outputs a ranked list of items from a catalog. In order to measure the performance of such an algorithm, we need to know what items would be relevant for each user. Gathering this information is costly, and sometimes even impossible. However, it is easy to collect user behaviors such as interaction with items, thus, content recommendation is commonly approximated to an interaction prediction task.

Interaction prediction is a self-supervised learning task where interactions are used as labels to predict, before being given as inputs to predict future interactions. What sets this task apart from other self-supervised tasks is the relational aspect of the information used; each interaction explicitly connects information with other interactions and elements at play (such as users or items). In addition, interaction order and temporality usually play an important role in explaining the observed behaviors.

LiMNet [1] is a simple Machine Learning model that is designed to process interactions in a causal way, leveraging both relational information and the order of the interactions. This model proved its performance at solving the task of botnet detection in IoT networks, and has been designed in a modular and adaptive way that makes it easy to employ for different tasks. Given these promising results and that it is designed to exploit precisely the specific information that makes interaction prediction challenging, we believe that LiMNet can be an interesting solution to the interaction prediction task.

1.2. Problem

The core research question for this work is the following:

"How will LiMNet perform on the task of interaction prediction on a user-item network?"

LiMNet has shown significant success and potential, it has only been tested on the task of botnet detection on IoT networks, with some additional results available for the task of cryptocurrency fraud detection [2]. Thus, the goal of this project is to apply and evaluate LiMNet on interaction prediction tasks, to assess the capabilities of the architecture across a wider variety of problems. Answering that question would help us towards the following two goals:

1. Exploring the range of applications of the LiMNet model.
2. Helping understand the success of state-of-the-art interaction prediction models with similar architectures.

In addition, this project implements and evaluates adaptations and changes to the model's architecture, attempting to further enhance its performance on the task of interaction prediction, but also to explore further the space of possibilities opened by the design of LiMNet .

1.3. Delimitations

This project focuses solely on predicting interactions for user-item networks. Thus, it does not cover tasks such as user/item label classification or link prediction between users or items separately.

1.4. Contributions

We list here the contributions of this project:

- We publish a framework for model evaluation on the user-item interaction prediction task.
- We present and evaluate LiMNet , a new embedding model taken from IoT botnet detection.
- We propose and test modifications to this model targeted at improving performance for the specific task of user-item interaction prediction.
- We reproduce and evaluate a state-of-the-art model as a baseline.

1.5. Ethics and Sustainability

The progress of Machine Learning applications, which allows for leveraging big data sources at the expense of large infrastructure costs, increases the risk of inequalities by increasing the power given to the biggest institutions. However, for this project, that risk is mitigated thanks to three aspects of the LiMNet architecture. Firstly, the big selling points of this architecture are its scalability and lightweight aspect, both elements that contribute to reduce the entry cost to run such a model. Secondly, there is ongoing research to adapt this architecture into a decentralized and collaborative variant [3]. That variant may allow communities to leverage the model using distributed resources. Lastly, this project is public and thus easily accessible for legislators and law enforcers. There is an ever-increasing need to push the legislation on the use of new technologies, and research allowing legislators to take informed decisions about these subjects that still contain a lot of unknowns is valuable.

The reduced cost of computing power is also subject to potential environmental impacts. It is, however, still unclear if increasing energy efficiency leads to an actual energy saving in the long run [4].

Chapter 2
Background

This chapter provides the background for the project. In Section 2.1, we provide an overview of link prediction and the classical solutions for the problem. In Section 2.2, we further develop the concept of graph embedding and some common methods to create them. Then, in Section 2.3, we discuss the addition of a time dimension in graph-shaped data and the way it can be exploited, followed by a presentation of cross-RNN architectures in Section 2.4. Finally, we present the model of interest for this work in Section 2.5 and why we believe that it is a relevant addition to the task of link prediction.

2.1. User-Item Link Prediction

Content personalization can commonly be represented as a link-prediction problem in a user-item graph. In such a graph, each user and each item is associated with a node. An item can be any kind of information or content the user is interested in, such as web pages, music tracks, or items in an e-commerce catalog. For each interaction a user has with the system, it registers as an edge in the graph. The goal of the personalization system is to find which items are the most relevant for a given user at a given time, which is the same as predicting which interactions could be added next in the graph. Note that the relevance of an item is highly contextual; for example, Christmas songs are very relevant around December and much less in July.

Two different ways to approach that problem are to use graph analytics, or to use features. By measuring how close each item is to the user in the graph through graph analytics methods, we get recommendations based on the user's past interactions, and on the interactions users with a similar history had. For example, if two users listened to the same songs, there is a higher chance that the second user will also like the other songs that the first user likes. Research in graph theory has provided us with a range of different ways to compute closeness between two nodes, such as measuring the shortest path connecting them, how many neighbors they share, or how exclusive their common neighbors are.

The other approach exploits additional information we get in addition to the relationship between users and items. Most real-world systems provide rich information about the nature of each interaction, users, and items. For example, in a music streaming service, a song can have a length and a genre, while a user can have an age. We call these information features, and exploiting them is the core of Machine Learning. This approach doesn't make suggestions based on the users' past interactions but instead will suggest similar results to users with similar attributes. To continue on the music streaming service example, we could identify relationships between the different populations of users and items, such as audience of young parents being more likely to be interested in nursery rhymes than teenagers.

The challenge is then to meld both approaches. Lichtenwalter et al. proposed to approach link prediction as a supervised Machine Learning problem where the objective is to predict if an edge will exist in the future between a given pair of user and item. To include the relational data in their model, they add some of the closeness metrics from graph analytics to the users and items features[5]. This setup invites to exploit the graph-based data not directly to predict the result but to build relevant features to enhance existing feature-based solutions. Instead of looking for the desired property in the structure of the graph, this approach will try to summarize the structure into rich representations compatible with machine learning algorithms.

All the previously mentioned methods present one main drawback: each time a user wants to be predicted an item, the score for each item regarding that user must be evaluated. This constraint makes it impossible to scale the solutions to large pools of items. To limit the number of comparisons, a solution is to create a high-dimensional representation of the users and items separately and use simple proximity functions on these embeddings as a scoring function. This spatial representation allows to reduce the problem to a nearest neighbor search for which scalable solutions have been found.

2.2. Graph Representation Learning

The task of learning high-level representation from graph data is called Graph Representation Learning, abbreviated as GRL . GRL is a broad family of machine learning approaches that capture the complex non-Euclidean structure of graphs into low-dimensional Euclidean representations (i.e., numerical vectors), enabling its use by downstream ML methods. It can be used to classify graph structures such as protein graphs, to capture information

from a subgraph, for example, by creating subgraph representations from knowledge graphs to feed into Large Language Models, and more commonly, to create embeddings for nodes in a graph. These embeddings must capture information about the node's features but also about the context of the node in the graph, which is typically defined as the neighboring nodes and their respective features and context.

2.3. Dynamic Graphs

Most of the information we get from networks is dynamic, especially for user-item interaction networks, where each interaction usually happens at a given time. Yet, when dealing with relational data, this temporal dimension is often disregarded to limit the complexity of the problem or simplified as a mere feature of the interaction. However, temporal data constitute a unique kind of information, allowing for the exploitation of the causal relationships between the different interactions.

Causality is the idea that causes will have consequences in the future. It becomes especially critical when studying phenomena that can spread through the networks, such as diseases, information, or trends. In such settings, each interaction can be the cause for a new state in the interacting nodes, requiring a different treatment for the same node at different times. While this concept is very intuitive for us, it is not the case for common GRL techniques described in Section 2.2 that let the information spread along the graph regardless of the order in which they are created.

In their review of dynamic network[6], Zheng et al. explain two ways temporal information is commonly included into graph data. The first one considers a series of snapshots of the graph at successive timestamps. The second one, called Continuous-Time Dynamic Graph (CTDG) , records every edition to the graph as an event, associated with a timestamp. A typical event in a CTDG is an edge addition or deletion. For this work, the focus is on CTDG with all events being punctual interactions. We call such networks temporal interaction networks. These networks have the benefit of representing the reality of a lot of systems in a completely faithful way because they continuously account for the new modifications. However, the structure of the graph is blurry as each interaction corresponds to a point-in-time edge that is deleted as soon as it appears. Because of this, we tend to approach such graphs as a stream of interactions rather than a structured network.

A popular approach to leverage temporal data when creating node embeddings is to maintain a memory of the embeddings and update them as interactions are read. One of the building blocks for this approach is DeepCoevolve [7], a model for link prediction that uses two components: a cross-RNN (detailed further in Section 2.4) to update the representation of the users and items, followed by an intensity function to predict the best match for the user at every given time t. Following DeepCoevolve, other cross-RNN models have been proposed with notable performance upgrades.

JODIE [8] builds upon DeepCoevolve by adding a static embedding component to the representation, using the Cross-RNN part to track the users' and items' trajectories. It then employs a neural network layer to project the future embedding of each node at varying times, an operation that replaces the intensity function in DeepCoevolve.

DeePRed [9] is another approach building on top of DeepCoevolve, this time with the aim to accelerate and simplify the training by getting rid of the recurrence in the cross-RNN mechanism. To achieve this, the dynamic embeddings are computed based on static embeddings, effectively getting rid of the recurrence by never reusing the dynamic embeddings for further computations. The lack of long-term information passing is compensated by the use of a sliding context window coupled with an attention mechanism to best identify the meaningful interactions.

2.4. Cross-RNN

The key mechanism for all the aforementioned models is called cross-RNN, where RNN stands for Recurrent Neural Network. A RNN is a neural network with the specificity of processing sequential data, passing an internal memory embedding between each step of the sequence of inputs. Formally, a RNN layer is defined as

$$\boldsymbol{o}(\boldsymbol{i}_t) = f(\boldsymbol{i}_t, \boldsymbol{h}_{t-1}) \tag{1}$$

$$\boldsymbol{h}_t = g(\boldsymbol{i}_t, \boldsymbol{h}_{t-1}) \tag{2}$$

Where t stands for the time step of the input $\boldsymbol{i}_t$. $\boldsymbol{o}(\boldsymbol{i}_t)$ marks the output of the layer and $\boldsymbol{h}_t$ represent the memory of the layer after receiving the input $\boldsymbol{i}_t$. The functions f and g can vary depending on the nature of the RNN , but they will rely on weights, tuned during the model training. Popular RNN architectures try to keep a memory of long-term knowledge. Typically, the

Long-Short Term Memory (LSTM) architecture maintains two distinct memories, a short-term one and a long-term one. The Gated Recurrent Unit (GRU) architecture iterates over LSTM by simplifying it, removing one of the two memories while keeping the gating mechanism. In practice, both approaches perform significantly better than the naïve RNN implementation, with GRU achieving comparable performances to LSTM , in spite of its reduced cost.

A Cross-RNN layer is slightly different. Instead of keeping track of a single memory embedding $\boldsymbol{h}_t$, it maintain a memory for all nodes in the graph $\boldsymbol{H}_t = (\boldsymbol{h}_t^u)_{u\in\mathbb{U}} \cup (\boldsymbol{h}_t^i)_{i\in\mathbb{I}}$, where $\mathbb{U}$ and $\mathbb{I}$ are the sets of users and items in the graph. For each interaction $(u, i, t, \boldsymbol{f})$ the memory is updated as follow:

$$\boldsymbol{h}_t^u = g^u(\boldsymbol{h}_{t-1}^u, \boldsymbol{h}_{t-1}^i, t, \boldsymbol{f}) \tag{3}$$

$$\boldsymbol{h}_t^i = g^i(\boldsymbol{h}_{t-1}^i, \boldsymbol{h}_{t-1}^u, t, \boldsymbol{f}) \tag{4}$$

And for all other users and nodes, the memory is carried over.

$$\boldsymbol{h}_t^v = \boldsymbol{h}_{t-1}^v \qquad \forall v \in \mathbb{U} \setminus \{u\} \cup \mathbb{I} \setminus \{i\} \tag{5}$$

Where u is the user interacting with item i at time t with feature $\boldsymbol{f}$, and g^u and g^i are tunable functions, comparable to the function g in Eq. 2. LSTM and GRU cells are designed for classical RNNs can be used for cross-RNN, the only modification being the memory management external to the cell.

The main benefit of cross-RNN architectures is that conservation of causality is granted by design. It comes, however, with a cost: the input of a cross-RNN model is sequential and cannot be made parallel. This cost is nonetheless mostly an issue for the training of the model, because processing one input in inference does not require passing through the entire sequence.

2.5. LiMNet

LiMNet is a cross-RNN model designed to optimize the memory utilization and computational speed at inference time. In the original paper[1], LiMNet is designed as a complete framework for botnet detection in IoT networks, with four main components: an input feature map, a generalization layer, an output feature map, and a response layer. We will, however, for the purpose of this work, consider LiMNet as a graph embedding module. Because the input feature map, output feature map, and the response layer are task-

dependent, the denomination “LiMNet” in this present work will designate only the generalization layer from this point.

LiMNet , as a graph embedding module, is a straightforward implementation of a cross-RNN module. This simplicity in the design brings two main benefits: first, LiMNet is very cheap to run at inference time, with a memory requirement linear in the number of nodes in the network. Secondly, it is flexible to node insertion and deletion. If a new node is added to the graph, its embedding can be computed immediately, without the need to retrain the model. Node deletions are even easier to handle; all it takes is to delete the corresponding embedding from memory. In practice, LiMNet has already proven its potential on the task of IoT botnet detection[1] and fraud detection[2], it is thus expected that it could yield satisfying results for link-prediction, while requiring fewer resources than state-of-the-art solutions.

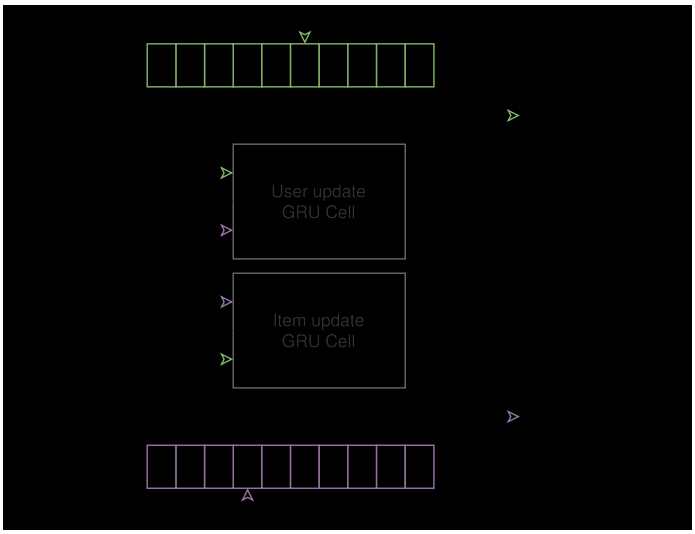

Figure 2.1: Architecture of LiMNet . User embeddings are indicated in green and items embeddings in purple.

Chapter 3
Method

In this chapter, we detail the experiments conducted throughout this work. First in Section 3.1, we introduce the datasets used in this work. Section 3.2 details the framework developed to conduct the experiments in a fair and controlled environment. Next, we present in Section 3.3 the various adaptations proposed for LiMNet to solve the task of link-prediction. Finally, we discuss in Section 3.4 exploration we conducted with the baselines.

3.1. Datasets

We use three public datasets in this project, all directly taken from the Stanford Large Network Dataset Collection (accessible at snap.stanford.edu/jodie/#datasets). More specifically, these datasets were created by Kumar et al. in [8] and have been reused a large number of times since then, to the extent that they have become de facto standard benchmarks for interaction network predictions.

- **Wikipedia edits:** This dataset gathers edits on Wikipedia pages over the course of a month. It is made of the 1,000 most edited pages during the month and the 8,227 users who edited at least 5 times any of these pages. In total, the dataset records 157,474 edits.

- **Reddit posts:** This dataset was built similarly to the Wikipedia dataset. The interactions on this dataset are 672,447 posts published by the 10,000 most active users on the 1,000 most active subreddits, over the course of a month. In total, this dataset records 672,447 interactions.

- **LastFM songs listens:** This dataset records music streams of the 1,000 most listened to songs on the LastFM website. These streams are performed by 1,000 users throughout one month and result in 1,293,103 total interactions.

The initial publication also included another dataset compiling user interaction with massive open online courses (MOOCs). This dataset records interaction events performed by 7,047 students on 97 courses. However, we

	Wikipedia	Reddit	LastFM
Users	8,227	10,000	1,000
Items	1,000	1,000	1,000
Interactions	157,474	672,447	1,293,103
Unique edges	18,257	78,516	154,993

Table 3.1: Details of the datasets.

decided to set it aside because we considered that it doesn't constitute a relevant use case for interaction prediction, as we expect users connecting to MOOC platform to already know on which course to work on.

The Table 3.1 summarizes the characteristics of the datasets. We can see that the main difference between the Wikipedia and the Reddit datasets is the density. They both have a similar number of users and items, but in Reddit there are about four times more connections between them than in Wikipedia. In LastFM the density is almost 20 times higher compared with Reddit. Note also that the balance between users and items is perfectly respected, compared with the two other datasets.

3.2. Experimental framework

Evaluating embedding models is a complex task because it requires accommodating a wide variety of input and output shapes, along with diverse training and inference procedures, while still ensuring the fairness of the evaluation between the different methods.

This complexity blooms with temporal graphs because there are different ways to approach them. One model can be approaching a temporal graph as a series of static graphs, another one can approach it as a time series [6] , and yet another one could try to maintain a dynamical representation of the graph on the fly. None of these approaches is inherently better or worse than the others, and they can all open up different design opportunities.

The implementation we came up with is publicly available on GitHub under the following link: https://github.com/mazerti/link-prediction.

In this section, we will detail the design decisions that led to our final evaluation framework. We grouped these decisions into four categories: Preparation of the data, Batching strategy, Training and evaluation loops, and comparison of the embeddings.

3.2.1. Preparation of the data

The datasets provide us with three types of information for each recorded interaction: the identifiers of the interacting user and item, the timestamp at which the interaction took place, and a set of features providing additional information about the interaction. Most of the time, the models tested use only the identifiers and the implicit order of the interaction. Thus, the framework will always provide as inputs the IDs of the user and item interacting in the order the interactions happen.

In addition, the framework can add features to the inputs. These features can be requested either by the user through the configuration file or directly by the model's implementation during the model's initialization. This second option allows for seamless use of models relying on custom features without the requirement to manually request the features each time the model is used. This is especially relevant for time information, because each model can have a different use of the timestamps. A common usage is to use the time delta between successive interactions of the same user. This information would be expensive to compute at inference because it would require keeping track of the timestamp of the last interaction each user has performed at any time step. Pre-computing it as a feature, on the other hand, is much more convenient because we have access to all the interactions at once, allowing for the computation of the time deltas in a simple query operation.

3.2.2. Batching Strategy

As pointed out by previous work [8], [9], temporal interaction comes with a tradeoff regarding the ability to leverage parallelism for training. Kumar et al. proposed for their model JODIE an elaborate batching strategy based on the structure of the graph [8], and Kefato et al. removed the recursions from their model DeePRed by approximating the dynamic embeddings with static ones [9].

Inspired by the original LiMNet proposition [1], we decided instead to slice the data into sequences of fixed size. The idea is that big enough sequences could be a good enough approximation of the actual sequence of all the interactions. While each sequence still needs to be processed in order, several sequences can be processed in parallel, speeding up the training.

3.2.3. Training and evaluation loops

The on-demand feature preparation discussed in Section 3.2.1 ensures that each model can access the inputs that it requires. Unfortunately, the models' outputs also present structural discrepancies. Since this work's scope is limited to embedding models, all models' outputs should be fixed-size embeddings for either users or items. Some models, though, come with their own loss functions based on internal states. Accessing the right outputs to compute either the loss function or the metrics from the embeddings is therefore not something that can be managed identically for all models.

Because of this limitation, we decided to tie the evaluation logic of the models to the models' implementations. This includes going over a batch of interaction sequences, running the predictions, computing the loss, back propagation, updating the model memories, and producing measurable embeddings. However, we made sure to standardize all measures with fixed functions designed independently from the models with the sole purpose of measuring embeddings for the task of link prediction. In addition, the training and evaluation loops are managed in a unified way, limiting the risk of bugs occurring due to errors in code reproduction. These loops include going over the batches, reporting the results, and iterating over the epochs.

3.2.4. Comparison of the embeddings

The last challenge in the implementation concerns the embeddings. While we want to create embeddings to synthesize the information, it is not the actual end goal of the system. The end goal is to rank the items for a given user, and, if the model is performant, to rank the item that will interact with the user high on the list. There are several ways to convert the embeddings into a ranking. Because this is not the focus of this work, we decided to use a simple approach: we rank the item embeddings based on their proximity to the user embedding. The framework, however, uses two separate approaches to compute the proximity between embeddings to best accommodate the diversity of models tested.

The first one is computed as the dot product of the normalized embeddings, which is equivalent to the cosine of the angle formed by the two embeddings with the origin of the embedding space.

$$\text{dot\_product\_score}(\boldsymbol{e}^{\text{user}}, \boldsymbol{e}^{\text{item}}) = \frac{\boldsymbol{e}^{\text{user}}}{\|\boldsymbol{e}^{\text{user}}\|} \cdot \frac{\boldsymbol{e}^{\text{item}}}{\|\boldsymbol{e}^{\text{item}}\|} \tag{6}$$

The higher the dot-product score between an item embedding and the target user embedding, the closer these embeddings will be, therefore, the item should be ranked higher for that user.

The second proximity used is the L2 distance, a generalization of geometric distance to k-dimensional spaces.

$$\text{L2\_score}(\boldsymbol{e}^{\text{user}}, \boldsymbol{e}^{\text{item}}) = \sqrt[k]{\|\boldsymbol{e}^{\text{user}} - \boldsymbol{e}^{\text{item}}\|^k} \tag{7}$$

For this score, lower values will indicate closer embeddings and be ranked higher.

For all experiments in the present work, the performances are measured with both scoring methods, and only the highest measured value among the two is reported.

3.2.5. Code maintainability

For exploratory research projects such as this one, it could be beneficial to write the whole codebase from scratch rather than reusing an existing codebase. Doing so removes from the research process the cruft of dependency management, the need to understand the details of the implementation, as well as the necessity to comply with a previous framework. Starting from scratch also allows to approach problems from a different angle, and generally lets the researcher focus on challenges emerging from the novelty. However, it is crucial to be able to reproduce experiments and to re-use existing models that can be used as baselines or as a base for further developments.

Thus, this framework has been developed to be easy to understand and either build upon or reproduce. To reach this goal, two lines have been followed through the development process: thorough documentation and a functional programming approach. The systematic documentation of every function in the framework should help future researchers understand the details of the implementation faster, either for reusing it or to reproduce its behavior in a new experimental context. With the same goal of simplicity of understanding, the state has also been gathered as much as possible into a single location: the `Context` class. This class acts as a simple store for all stateful parts of the framework, making them never more than one variable away, wherever it is called from. In addition, the framework has been written following a functional programming-inspired style, always favoring pure

functions for their conceptual simplicity and consistency. Unfortunately, this functional approach couldn't be applied to every part of the program, notable exceptions are the PyTorch modules that had to be implemented in an object-oriented way to accommodate PyTorch's framework.

3.3. Adaptations of the LiMNet architecture

This work's primary goal was to test how the LiMNet model would perform for the link-prediction task. However, as discussed in Section 2.5, the implementation we use is stripped down of its input and output maps, as well as the response layer used in the original paper to fit the specific needs of the task of IoT botnet detection.

3.3.1. Loss functions

We also had to adapt the loss used to train LiMNet , because, unlike botnet detection, link-prediction isn't a classification setting, so we couldn't use cross-entropy Loss as the original model did. Instead, we decided to use a mix of two losses. The first is an objective loss to minimize the distance between the embedding of the interacting user and item, which is calculated using the mean squared error for the embeddings or their dot product to 1. And the second is a regularization loss to maximize the information retention by maximizing the distance between different users and between different items. This loss is computed as follows:

$$L_{\text{reg}} = \boldsymbol{U}\boldsymbol{U}^T + \boldsymbol{I}\boldsymbol{I}^T \tag{8}$$

Where $\boldsymbol{U}$ and $\boldsymbol{I}$ are respectively the matrix containing all the users' embeddings and the matrix containing all the items' embeddings.

In addition to this simplification, we proposed three modifications to enhance the model: adding time features, normalizing the embeddings, and stacking several LiMNet layers.

3.3.2. Addition of time features

While LiMNet takes advantage of the order of the interaction to propagate information in a causal way, it doesn't use the actual timestamps to compute the embeddings. One of our assumptions was that this would lead to a loss of relevant information that could otherwise have been useful to predict the best item. In order to check that assumption, we created time features to provide the model with information about when the interaction happened.

We specifically intended to capture cyclic patterns in the user behaviors, such as week/weekend or day/night differences in behaviors.

Unfortunately, our datasets only provide relative timestamps that obfuscate the exact time and day of the interactions, so we had to approximate these patterns by using a frequency decomposition of the timestamps. Specifically, we use the following two features to capture a temporal pattern:

$$\cos\left(\frac{2\pi}{\Delta}t\right), \sin\left(\frac{2\pi}{\Delta}t\right) \tag{9}$$

Where t is the timestamp of the interaction, and Δ is the duration of the pattern we want to capture (a day or a week) in the unit of the timestamps. This aims at providing the model with a time representation that is more compatible with its machine learning components, which tend to fail to extract patterns from one-dimensional values.

3.3.3. Normalization of the embeddings

An efficient way to compute the dot-product scores (Eq. 6) is to normalize all the embeddings to the unit sphere. While doing this, we realized that it could also be applied to the embeddings in LiMNet ‘s memory. This way, the cross-RNN mechanism is also fed with normalized embeddings as inputs. Our hopes were that this way, the model would be more focused on encoding information through the angles between the embeddings rather than through their amplitudes.

Another benefit of this method is that it prevents the embeddings from collapsing to 0. While the regularization loss is meant to prevent embeddings from all converging to the same value, it actually only makes sure that the embeddings are not aligned, therefore, 0 remains a potential convergence point. Keeping the embeddings normalized at any time is thus a good solution to this issue.

Our experiments yielded significantly better results with embedding normalization, so we decided to use this modification by default for all the experiments conducted with LiMNet on this project.

3.3.4. Stacking several LiMNet layers

The last improvement to LiMNet that we have tried was to stack several layers of the LiMNet architecture on top of each other, effectively turning it into a deep recurrent neural network. Figure 3.1 illustrates the architecture

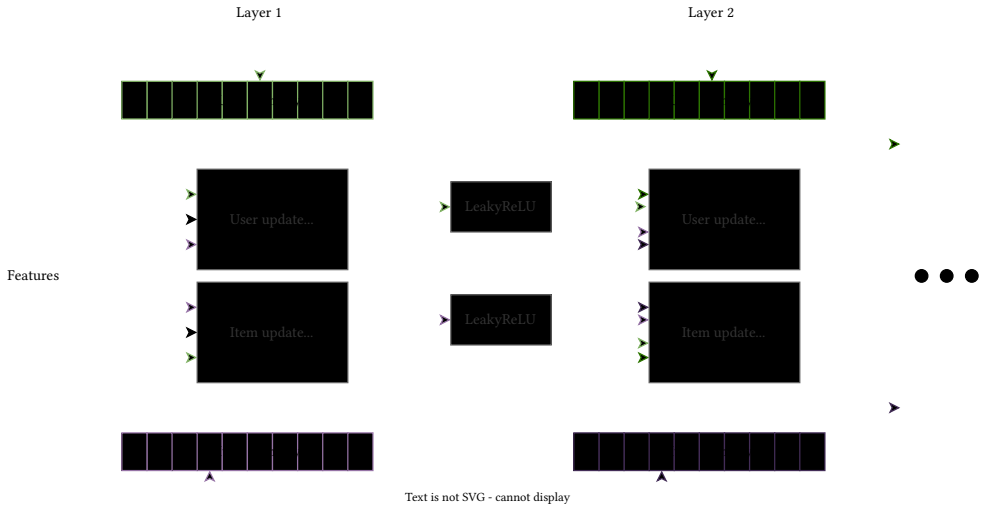

Figure 3.1: Architecture of LiMNet with two layers.

of this new model. The leaky ReLU functions inserted between each layer aim to add non-linearity and increase the expressiveness of the model.

3.4. Baselines

We evaluated the performance of LiMNet against 2 other baselines: static embeddings and Jodie.

We trained static embeddings for each user and item, with the same loss functions that we described in Section 3.3.1 for LiMNet . This baseline is oblivious to the relational and temporal information contained in the data. It is also inconvenient to deploy for real real-world application because it requires being entirely re-trained to account for any new information, such as new interactions, new users, or new items.

The other baseline, Jodie, described in [8], is built upon the same core of cross-RNN embeddings as LiMNet , but presents three major differences. First, in addition to the cross-RNN dynamic embeddings, Jodie uses one-hot representations of the users and items to create the final embeddings. Secondly, Jodie exploits the time delta between two interactions of a user throughout a projection operation that tries to anticipate the embeddings' trajectory. Lastly, the model is trained with a dedicated loss function that ensures that the embeddings won't change too radically as a consequence of an interaction.

We identified two differences between our implementation of Jodie and the original proposition: the absence of the t-batch algorithm, replaced by fixed-length sequences, and the absence of interaction features, ignored for simplicity. Compared with the static embeddings, Jodie doesn't need to be re-trained to acknowledge new interactions, but it still can't deal dynamically with user or item insertion or deletions.

Chapter 4
Experiments

In this chapter, we present the experiments performed throughout this project. The first section covers how we tried to improve on the core of the limnet model. Then we evaluate the impact of the temporal information on the models.

To give some context, Figure 4.1 shows the performance of the two models we tested, LiMNet and Jodie. It stands clearly that Jodie is outperforming LiMNet by a wide margin, regardless of the dataset used.

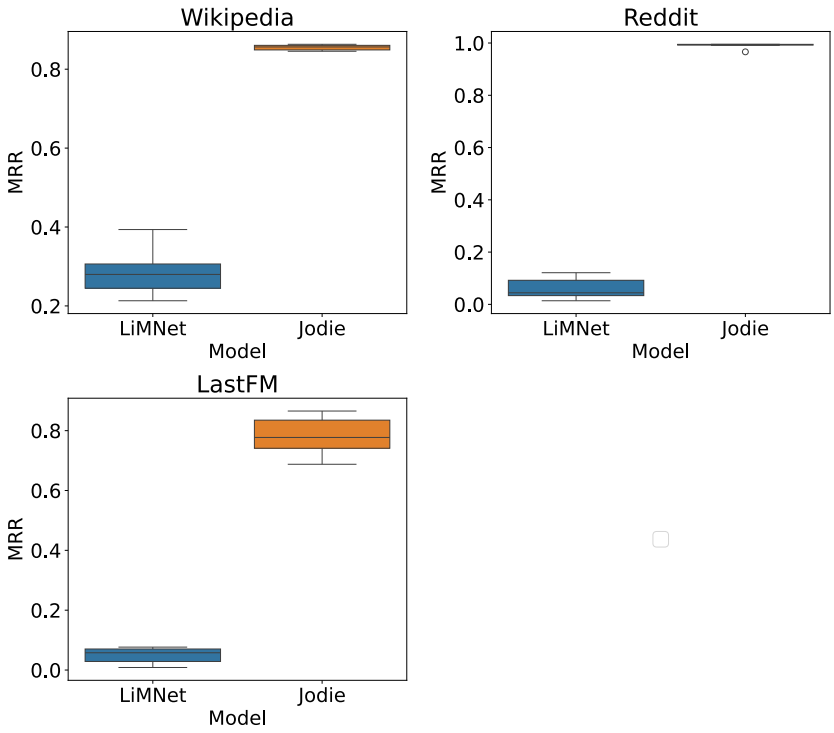

Figure 4.1: Comparison of LiMNet and Jodie models.

4.1. Improvements on limnet

Three different modifications have been tried with the hope of reducing the gap between LiMNet and Jodie. The three following subsections present these modifications and their experimental results.

4.1.1. Adding time features

The most important specificity of the models tested lies in their ability to leverage temporal relationships between users and items. However, LiMNet only exploits the order in which interactions happen, with no consideration for their exact time.

To resolve this absence, this experiment tries to pass the exact interaction timestamps as features to the LiMNet model. More specifically, the features passed to the model seek to capture cyclic patterns in the interactions. Unfortunately, the datasets only provide relative timestamps that do not carry the precise time and day of the interactions. We thus approximated these patterns through a frequency decomposition of the following form:

$$\cos\left(\frac{2\pi}{\Delta}t\right), \sin\left(\frac{2\pi}{\Delta}t\right) \tag{10}$$

Where t is the timestamp of the interaction, and Δ is the duration of the pattern we want to capture (a day or a week) in the unit of the timestamps. Using cosine and sine is a trick that provides the model with easily comparable features compared to directly passing the timestamps as a feature.

Figure 4.2 Shows the impact of these features on the measured Mean Reciprocal Rank (MRR) . Adding features seems to reduce the variance of the model, but may have a negative impact on the average results. In any case, it seems clear that the model struggles to deal with these additional features in a meaningful way. This experiment thus constitutes a case against using such features as a way to enhance the LiMNet model for link prediction.

4.1.2. Normalizing the embeddings

Forcing the embeddings to lie on the unit sphere pushes the model to encode information through the angle the embeddings form with the space origin rather than through their amplitude. This is appropriate when optimizing the embeddings for the dot-product score Eq. 6. In this experiment, we tried to systematically normalize the embeddings after each interaction; this

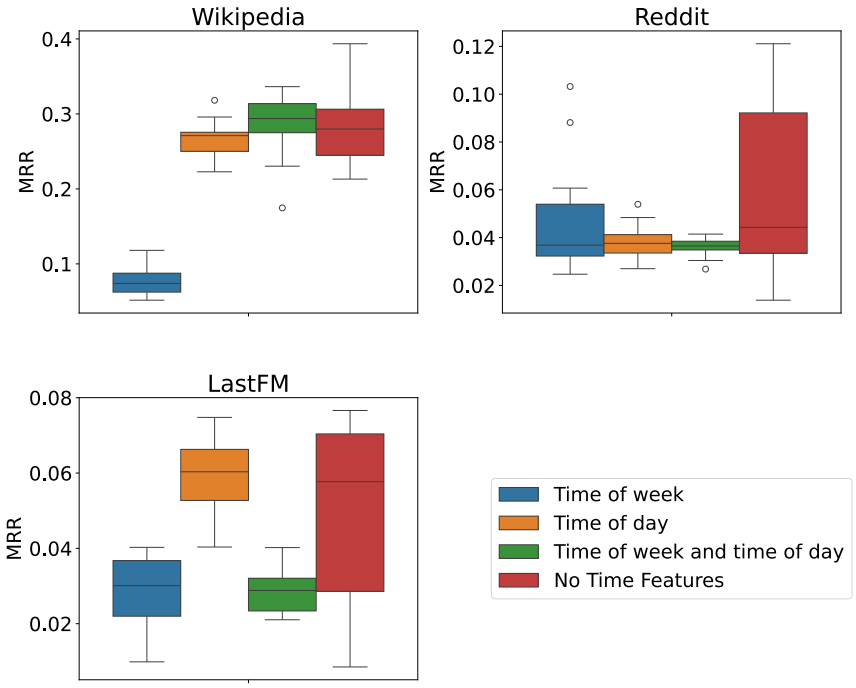

Figure 4.2: Performances of the LiMNet model with time features added.

way, not only the outputs but also the inputs of the cross-RNN are always normalized.

As you can see in Figure 4.3, this modification yields significantly better results. Thus, we apply it by default to all the other experiments presented in this report.

4.1.3. Stacking layers

This experiment tests whether stacking several layers of LiMNet could improve its results. Figure 3.1 illustrates the architecture of a stacked LiMNet model. In between layers, Leaky ReLU units have been added to add non-linearity and expressiveness to the model.

As shown in Figure 4.4, stacking more than two layers doesn't prove to increase the model performance. Another observation that can be made is that changing from one layer to two have a positive impact on the more

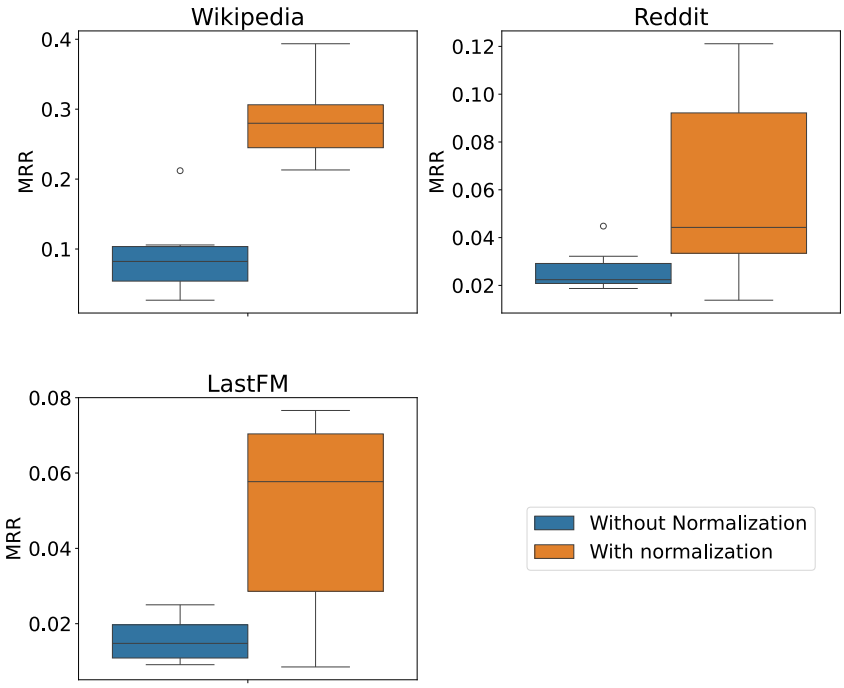

Figure 4.3: Performances of the LiMNet model with and without normalization of the embeddings.

complex dataset but tend to decrease the performance of the model for the simpler Wikipedia dataset.

4.2. Impact of the sequence size on the results

The last experiment aims to understand the impact of the temporal and sequential information on the performance of the models. It was performed by training and evaluating the models with sequence lengths of 16, 64, 256, and 1024, so that the model would only have access to up to this many successive interactions to perform their prediction.

The results displayed in Figure 4.5 show that both models perform, at best, as good with a bigger sequence length, which suggests a poor ability to exploit long-term information. However, except for LiMNet on Wikipedia, the sequence length seems to actually play only a marginal role in the

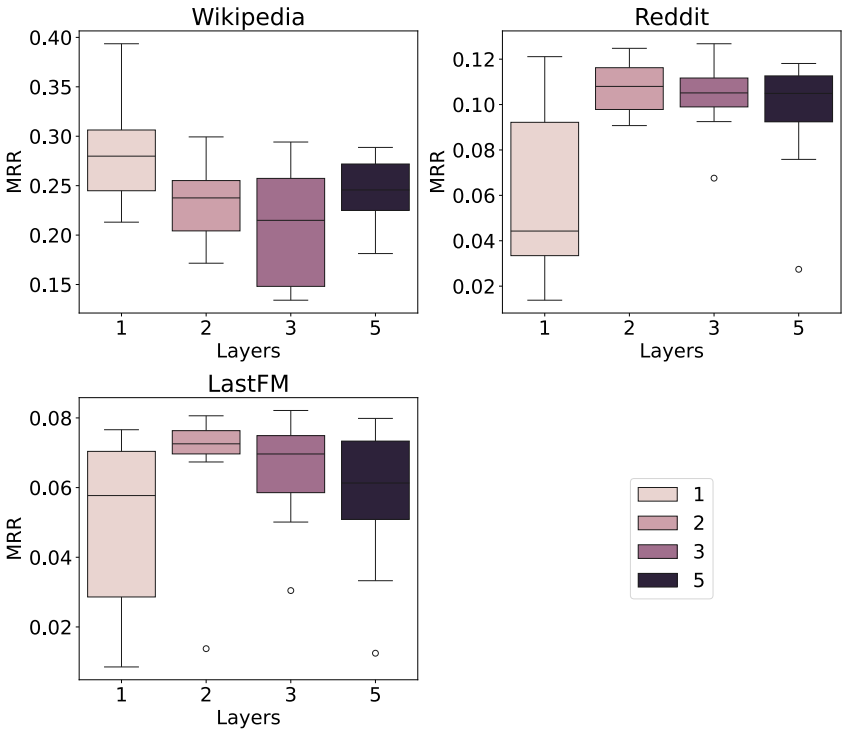

Figure 4.4: Performances of the LiMNet model with various numbers of stacked layers.

performance of both models, which suggests that they may not even alleviate temporal information as they were designed for.

Under the light of these observations, we assume that the models may have learned over short-term global popularity patterns instead of long-term local preferences. This behavior seems to make sense with regard to the nature of the datasets used, where recent interactions by other users are likely to provoke new interactions from other users. In addition, all these datasets contain only the 1,000 most popular items over a period of a month. That could have led to selecting mostly items whose popularity spiked around given times during the month before going down again, while the spotlight was turning towards other items.

Such a hypothesis could explain why our implementation of Jodie is reaching up to 60% higher MRR on LastFM compared with the original implementation. The main difference between our implementation and the

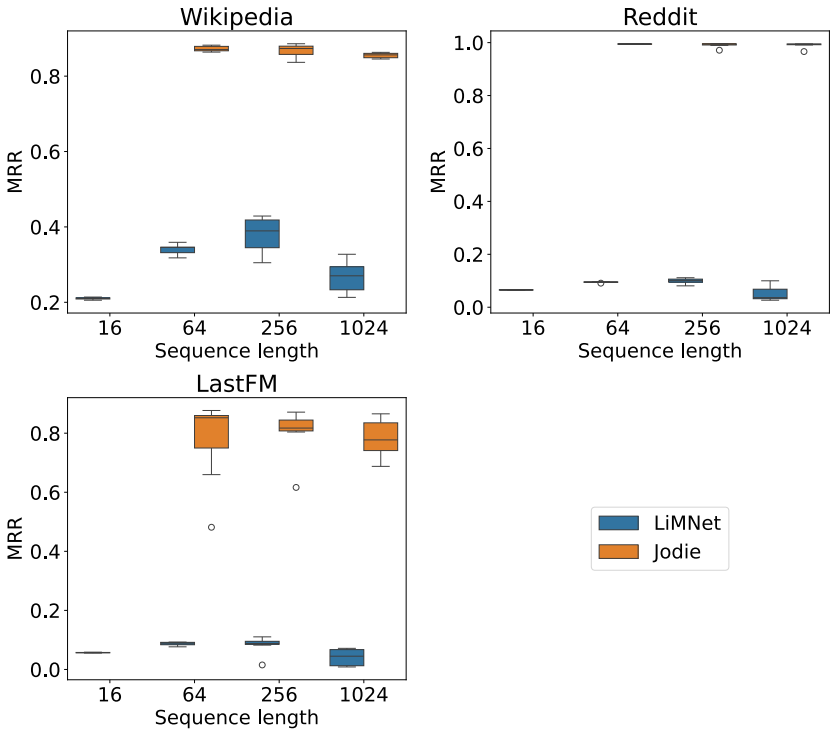

Figure 4.5: Effect of the sequence length on the models' performances.

one presented in the paper is the batching algorithm; in this project, interactions are processed in chronological order, while in [8] they are processed following the t-batch algorithm. The t-batch algorithm groups interactions with common items or users together, leading to a more localized information sharing mechanism.

Chapter 5
Conclusions

The goal of this thesis was to understand the potential of the LiMNet model for interaction prediction task on a user-item dynamical network. This research question led to the creation of an evaluation framework for this task, the evaluation of variants of the LiMNet model on this framework, along with re-implementing the Jodie model as a baseline. There are three key findings of this thesis: Firstly, LiMNet significantly underperform on the interaction prediction task compared with Jodie. Secondly, normalizing the embeddings throughout the cross-RNN mechanism improves the performance of the model. Lastly, the length of the interaction sequence processed by the models have little impact on the models results, hinting at short term global trends effect dominance over long term local preference behaviors.

5.1. Limitations

To keep the project's scope to a reasonable size, some questions had to be set aside during the research project, thus, the findings only concerns the task of user-item interaction prediction and doesn't address more general link-prediction tasks. In addition, this work focused on embedding creation with little consideration for how to most efficiently employ these embeddings. Another aspect that was left over during the research process was the performance tests, and more specifically the performance at inference. That aspect is one of the most benefit of the LiMNet architecture as demonstrated in [1], it is therefore good to keep in mind this limitation of the present project when assessing the potential of LiMNet .

5.2. Future Works

The experiment on sequence length discussed in Section 4.2 exposed a surprising and novel view on the underlying mechanisms that steer the behaviors recorded in the datasets used. Therefore, it would be very beneficial to try the models tested in this study on more suited datasets that do exhibits more complex long-term and local behaviors.

The other interesting gap we suggest to explore is the difference between the performances of the LiMNet model and the Jodie one, since both models share the same core, it is surprising to witness such a big difference in their capabilities. Exploring the design space that separates this models through an ablation study of the components of Jodie could bring more insight on which architectural decision generates such a performance leap.

5.3. Reflections

References

[1] L. Giaretta, A. Lekssays, B. Carminati, E. Ferrari, and S. Girdzijauskas, "LiMNet : Early-Stage Detection of IoT Botnets with Lightweight Memory Networks," in *Computer Security – ESORICS 2021 : 26th European Symposium on Research in Computer Security, Darmstadt, Germany, October 4–8, 2021, Proceedings, Part I*, in Lecture Notes in Computer Science. Springer Nature, 2021. doi: 10.1007/978-3-030-88418-5_29.

[2] Z. Cui, "Classification of Financial Transactions using Lightweight Memory Networks," 2022. [Online]. Available: https://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-321923

[3] L. Giaretta, A. Lekssays, B. Carminati, E. Ferrari, and S. Girdzijauskas, "Metasoma: Decentralized and CollaborativeEarly-Stage Detection of IoT Botnets."

[4] K. Gillingham, D. Rapson, and G. Wagner, "The Rebound Effect and Energy Efficiency Policy," *Review of Environmental Economics and Policy*, vol. 10, no. 1, pp. 68–88, 2016, doi: 10.1093/reep/rev017.

[5] R. N. Lichtenwalter, J. T. Lussier, and N. V. Chawla, "New perspectives and methods in link prediction," in *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, in KDD '10. Washington, DC, USA: Association for Computing Machinery, 2010, pp. 243–252. doi: 10.1145/1835804.1835837.

[6] Y. Zheng, L. Yi, and Z. Wei, "A survey of dynamic graph neural networks," *Frontiers of Computer Science*, vol. 19, no. 6, p. 196323, Dec. 2024, doi: 10.1007/s11704-024-3853-2.

[7] H. Dai, Y. Wang, R. Trivedi, and L. Song, "Deep Coevolutionary Network: Embedding User and Item Features for Recommendation," no. arXiv:1609.03675, arXiv, Feb. 2017. doi: 10.48550/arXiv.1609.03675.

[8] S. Kumar, X. Zhang, and J. Leskovec, "Predicting Dynamic Embedding Trajectory in Temporal Interaction Networks," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, in KDD '19. Anchorage, AK, USA: Association for Computing Machinery, 2019, pp. 1269–1278. doi: 10.1145/3292500.3330895.

[9] Z. Kefato, S. Girdzijauskas, N. Sheikh, and A. Montresor, "Dynamic Embeddings for Interaction Prediction," in *Proceedings of the Web Conference 2021*, in WWW '21. Ljubljana, Slovenia: Association for Computing Machinery, 2021, pp. 1609–1618. doi: 10.1145/3442381.3450020.

TRITA – EECS-EX 2024:0000
Stockholm, Sweden 2025
www.kth.se