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: June 04, 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 challenge for search engines, social media platforms, and streaming services, due to the need to model both relational structures and temporal dynamics. Many existing solutions address these two aspects separately, limiting their ability to fully capture user behavior.

In this work, we attempt to bridge that gap by evaluating Lightweight Memory Networks (LiMNet), a model designed to preserve causal relationships within sequences of temporal interactions. To assess its potential, we developed a benchmarking framework for user-item interaction prediction. We compared LiMNet against Jodie, a state-of-the-art baseline, across three real-world datasets: Wikipedia page edits, Reddit post submissions, and LastFM music streams. These datasets vary in scale and interaction patterns, providing a comprehensive testbed.

Our results show that while LiMNet offers advantages in efficiency and adaptability, it consistently underperforms compared to Jodie in predictive accuracy. Additionally, our findings hint at a consistent bias across all datasets toward short-term global popularity. This suggests that existing models may be overfitting to recent trends rather than learning long-term user preferences, highlighting a potential limitation in the current evaluation paradigms.

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, June 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, 19, 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 led to an overwhelming abundance of information, making it increasingly difficult to identify relevant and meaningful content. In response to this challenge, search engines and recommendation systems have become essential tools for filtering and navigating vast data landscapes.

Both systems share a common objective: to deliver information that is most relevant to the user. Content personalization has emerged as a particularly effective approach among the many techniques developed to achieve this. Rather than applying a one-size-fits-all filter, personalized systems adapt their output based on user-specific context such as search history, demographics, and past interactions.

Content personalization lies at the heart of recommendation systems and is highly effective in enhancing search engine performance. For instance, a search query for the term “football” should yield different results for a user interested in American football compared to someone seeking information about association football (soccer).

Content personalization can be represented as an algorithm that takes user-specific information as input and produces a ranked list of items from a larger catalog. These items can include any kind of content available to the user, such as songs in a music streaming service or web pages returned by a search engine.

Evaluating the performance of such algorithms typically requires knowledge of which items are genuinely relevant to each user. However, obtaining this information can be costly and, in some cases, unfeasible. On the other hand, user behaviors—such as clicks, listens, or edits—are easy to collect at scale. As a result, content recommendation is often approximated as an interaction prediction task, where the goal is to predict future user-item interactions based on past behavior.

Interaction prediction is a self-supervised learning task in which past user-item interactions serve first as labels and later as input features used to predict future interactions. Unlike other self-supervised tasks, it is distinguished by the inherently relational nature of the data: each interaction connects users and items, forming a web of dependencies that influence future behavior.

As a result, these interactions are often modeled as user-item networks to highlight the structural relationships between entities. Besides, the temporal dimension—including the order and timing of interactions—typically plays a critical role in accurately modeling and understanding user behavior over time.

LiMNet [1] is a simple machine learning model designed to process user-item interactions in a causal manner, leveraging both relational information and the order of interactions. The model has demonstrated strong performance in tasks such as botnet detection in IoT networks, and it is built in a modular, adaptive way that makes it easy to apply to other problems. Given these promising characteristics and its ability to exploit precisely the types of information that make interaction prediction challenging, LiMNet appears to be a compelling candidate for this task.

1.2. Problem

The core research question for this work is the following:

"Does LiMNet have the potential to compete against state-of-the-art models on the task of interaction prediction for user-item network?"

LiMNet has shown significant promise, yet its evaluation has been limited to botnet detection in IoT networks [1], with some additional results available for cryptocurrency fraud detection [2]. This project aims to apply and assess LiMNet in the context of interaction prediction, thereby evaluating the model's generalizability across a broader range of tasks. Addressing this research question supports two main goals: first, to explore the applicability of the LiMNet architecture to diverse domains, and second, to better understand the characteristics that drive the success of similar state-of-the-art interaction prediction models.

To this end, we develop a flexible evaluation framework supporting multiple models while ensuring fair and consistent comparisons. Additionally, we

implement and evaluate several architectural adaptations to LiMNet , aiming not only to improve its performance on interaction prediction but also to further investigate the design space enabled by its modular structure.

1.3. Limitations

One alternative approach to interaction prediction involves building a classifier to determine whether a given user is likely to interact with a specific item. This approach is excluded from the present work, which instead focuses on computing embeddings and generating ranked item lists.

Another limitation of this study is its strict focus on user-item networks. Interactions between entities of the same type, such as user-user or item-item connections, are not considered. For instance, tasks like friendship prediction on a social network fall outside the scope of this project. Consequently, each entity is strictly classified as either a user or an item, with no allowance for hybrid or hierarchical roles. And only one kind of interaction is considered, e.g. user listening to a song but not artist posting a song.

Furthermore, this work considers only punctual interactions—discrete events occurring at specific points in time. Continuous interactions are excluded, and due to data limitations, the duration of interactions is not modeled. Although interaction durations could offer meaningful insights into engagement or relevance, they remain outside the scope of this thesis.

1.4. Contributions

The main contributions of this project are as follows:

- Development and publication of a framework for evaluating models on the user-item interaction prediction task.
- Presentation and evaluation of LiMNet, an embedding model originally proposed for IoT botnet detection. Our evaluation shows that, in its current form, LiMNet does not match the performance of state-of-the-art baselines.
- Proposal and testing of architectural modifications to LiMNet aimed at improving its performance in the context of user-item interaction prediction.
- Reproduction and evaluation of Jodie, a state-of-the-art model for temporal interaction prediction. Interestingly, our implementation produced results that exceeded expectations.

- Identification of a possible structural bias in the benchmark datasets, suggesting they may favor global popularity trends over more complex, long-term behavioral patterns.

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

Interaction prediction can often be formulated as a link prediction problem within a user-item graph. In such a graph, each user and each item is represented as a node. Each user interaction with an item is registered as an edge in the graph. Predicting future interactions comes down to predicting which edges are likely to appear next.

In this project, we focus specifically on user-item interaction networks—systems in which all interactions occur between distinct user and item entities. Users and items play fundamentally different roles: users initiate interactions, while items are the targets. This introduces a structural asymmetry in the network, where interactions are always directed from a user to an item. In practice, this means that prediction tasks are framed from the user's perspective, with the goal of identifying the most relevant items they are likely to interact with next.

There are two primary approaches to this problem: one based on graph analytics and the other on feature-based methods. Graph analytics focuses on measuring the proximity between a user and various items in the graph, leveraging insights from the user's past interactions and the behaviors of similar users. For instance, if two users have listened to the same set of songs, one may likely enjoy the songs the other has listened to. Graph theory offers a variety of methods to compute closeness between nodes, including shortest

path lengths, the number of shared neighbors, or the exclusivity of those shared neighbors.

The second approach leverages additional information beyond the user-item relationship. Most real-world systems provide rich metadata about interactions, users, and items. For example, in a music streaming service, a song may include attributes like genre and duration, while a user may be characterized by age or selected language. These attributes are referred to as features, and utilizing them is central to machine learning approaches. Unlike graph-based methods, feature-based models recommend items by identifying similar users based on shared characteristics. Continuing the music example, the system might learn that songs with lyrics in Swedish are less likely to appeal to users that don't use Swedish as their primary language.

The challenge lies in integrating both approaches. Lichtenwalter et al. proposed framing link prediction as a supervised machine learning task, where the objective is to predict whether an edge will form between a given user-item pair in the future. To incorporate relational data into the model, graph closeness metrics are added to the user and item features[3]. This setup uses graph structure not as the direct basis for prediction but as a source of enriched features, enabling machine learning algorithms to work with abstracted representations of the graph.

Despite their strengths, these methods share a common drawback: for each user, a score must be computed for every possible item to generate a recommendation. This becomes computationally infeasible when dealing with large item catalogs. A common solution is to learn high-dimensional embeddings for users and items separately and then compute a similarity score, such as a dot product or distance measure, to rank items. This transforms the problem into a nearest-neighbor search, a well-studied task with many efficient and scalable solutions.

2.2. Graph Representation Learning

The task of learning high-level representations from graph data is known as Graph Representation Learning (GRL) . GRL encompasses a broad family of machine learning techniques aimed at transforming the complex, non-Euclidean structure of graphs into low-dimensional Euclidean representations

(i.e., numerical vectors), making them suitable for use in downstream machine learning tasks.

GRL methods can be applied to a wide range of problems. These include classifying entire graph structures, such as molecular graphs; extracting subgraph representations from knowledge graphs to be used in large language models; and, most commonly, generating node embeddings. These node embeddings must encode not only the intrinsic features of individual nodes but also the context in which they appear. This context typically includes neighboring nodes and their corresponding features and positions within the graph.

2.3. Dynamic Graphs

Much of the information generated in real-world networks is inherently dynamic, particularly in user-item interaction networks, where each interaction occurs at a specific point in time. Despite this, the temporal dimension is often ignored or simplified in order to reduce modeling complexity. Yet, temporal information carries unique value, enabling models to capture not just patterns but also the causal relationships between interactions.

Causality refers to the principle that actions can influence future outcomes. This is especially important when studying processes that propagate through networks, such as the spread of information, trends, or behaviors. In these scenarios, an interaction may change the state of the involved nodes, making it necessary to treat the same node differently depending on the time of observation. While humans intuitively understand these evolving patterns, many standard GRL techniques disregard temporal order, propagating information across the graph without regard to when interactions occurred.

In their review of dynamic networks [4], Zheng et al. outline two common approaches to incorporating temporal information into two models. The first involves representing the graph as a sequence of discrete snapshots, each corresponding to a specific time step. The second, known as Continuous-Time Dynamic Graph (CTDG) , treats each graph update as an event timestamped in continuous time, typically the addition or removal of an edge.

This work focuses on CTDG , where all events are modeled as punctual interactions, also referred to as temporal interaction networks. These networks

offer a faithful representation of many real-world systems, as they continuously track changes over time. However, they come with a challenge: the underlying graph structure becomes ephemeral, with each edge appearing only momentarily. As a result, these networks are often better understood as interaction streams rather than static or evolving graphs with persistent edges.

A popular approach for leveraging temporal data when generating node embeddings is to maintain a memory of embeddings and update them as interactions occur. One foundational model in this domain is DeepCoevolve [5], a model for link prediction that uses two components: a cross-RNN (further detailed in Section 2.4) to update user and item representations, and an intensity function to predict the likelihood of future interactions at any given time. Following DeepCoevolve, several cross-RNN-based models have been proposed, achieving notable performance improvements.

JODIE [6] extends DeepCoevolve by introducing a static embedding component alongside the dynamic one. The cross-RNN tracks the trajectory of users and items over time, while a neural projection layer predicts their future embeddings at different time steps, replacing the intensity function used in DeepCoevolve.

DePRed [7] builds upon DeepCoevolve with the goal of simplifying and accelerating training by removing recurrence from the cross-RNN mechanism. Instead, dynamic embeddings are computed directly from static embeddings, avoiding recursive updates. The absence of long-term memory is addressed through the use of a sliding context window and an attention mechanism that identifies and weights the most relevant past interactions.

2.4. Cross-RNN

The key mechanism underlying the models discussed in the previous section is known as cross-RNN, where RNN stands for Recurrent Neural Network. A RNN is a type of neural network designed to process sequential data by maintaining and updating a memory state across time steps. Formally, a RNN layer is defined as:

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

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

Here, t denotes the time step of the input $\boldsymbol{i}_t$. The function $\boldsymbol{o}(\boldsymbol{i}_t)$ produces the layer's output, and $\boldsymbol{h}_t$ represents the updated memory after processing $\boldsymbol{i}_t$. The functions f and g are parameterized transformations, typically involving learned weights. Popular RNN architectures, such as Long-Short Term Memory (LSTM) and Gated Recurrent Unit (GRU) , are designed to retain long-term dependencies more effectively than simple RNNs . LSTM maintains two types of memory (short-term and long-term), whereas GRU simplifies this structure by using a single memory vector with a gating mechanism. In practice, GRUs achieve performance comparable to LSTMs while being computationally less demanding [8].

A cross-RNN layer extends this concept by maintaining separate memory states for all nodes in a graph. At time t, the memory is represented as: $\boldsymbol{H}_t = (\boldsymbol{h}_t^u)_{u \in \mathbb{U}} \cup (\boldsymbol{h}_t^i)_{i \in \mathbb{I}}$ where $\mathbb{U}$ and $\mathbb{I}$ denote the sets of users and items, respectively. For each interaction $(u, i, t, \boldsymbol{f})$, the memory states of the involved user u and item i are updated according to:

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

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

For all other nodes $v \in (\mathbb{U} \setminus \{u\}) \cup (\mathbb{I} \setminus \{i\})$, the memory remains unchanged.

Here, g^u and g^i are node-type-specific update functions analogous to g in a standard RNN . LSTM and GRU cells can be used within the cross-RNN framework, with the key distinction that memory management is handled externally to the cell.

The primary advantage of cross-RNN architectures is their inherent preservation of causality. Since updates depend strictly on past states, the model respects the temporal order of interactions by design. However, this sequential nature introduces a limitation: cross-RNN models cannot be parallelized over the sequence during training. Fortunately, this constraint primarily affects training efficiency, during inference, each interaction can be processed independently, making the approach practical for real-time applications.

2.5. LiMNet

LiMNet is a cross-RNN model designed to optimize memory utilization and computational efficiency during inference. In its original formulation [1],

LiMNet is part of a comprehensive framework for botnet detection in IoT networks. The framework includes four main components: an input feature map, a generalization layer, an output feature map, and a response layer. For the purposes of this work, however, we consider only the generalization layer. Since the other components are task-specific, the term "LiMNet" in this thesis refers exclusively to the generalization layer.

As a graph embedding module, LiMNet provides a straightforward implementation of a cross-RNN mechanism, Figure 2.1 proposes a visualization of its architecture. This simplicity offers two main advantages. First, LiMNet is highly efficient at inference time, with memory requirements that scale linearly with the number of nodes in the network. Second, it offers strong flexibility in handling dynamic node sets: when a new node is introduced, its embedding can be computed immediately without retraining the model. Deleting a node is even simpler: its embedding can just be removed from memory without consequences for the other embeddings.

LiMNet has already demonstrated promising results in tasks such as IoT botnet detection [1] and cryptocurrency fraud detection [2]. Given its design and prior success, it is a compelling candidate for link prediction tasks, particularly in settings where computational efficiency is a key constraint.

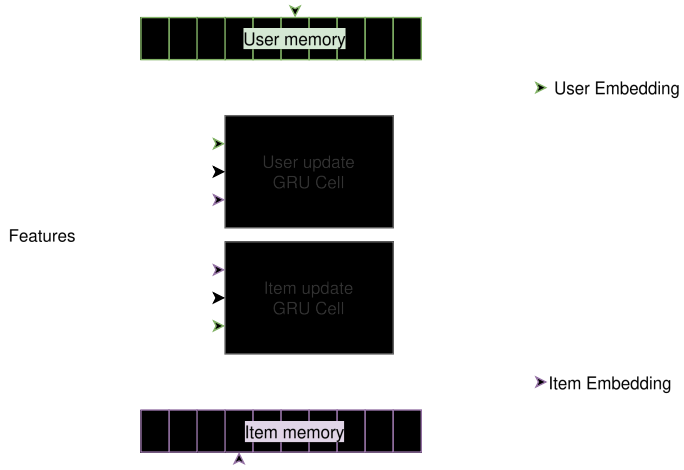

Figure 2.1: Architecture of LiMNet.

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 [6] 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 [4] , 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 [6], [7], 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 [6], and Kefato et al. removed the recursions from their model DeePRed by approximating the dynamic embeddings with static ones [7].

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.

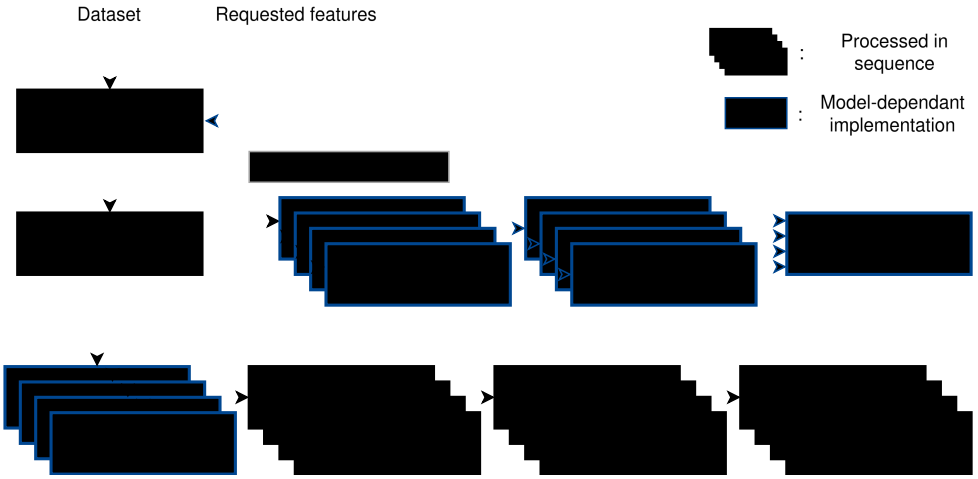

Figure 3.1: Schema of the evaluation framework. Blue indicates that the implementation is tight to the model evaluated.

3.2.3. Evaluation and Training loop

Designing a framework that fits any model for a given task is a very challenging task, because the models have been designed with different formulations of the problems in mind. A first significant difference concerns the forms of the inputs, as explained in Section 3.2.1, the framework takes the decision to use the features to accommodate with these discrepancies. A second source of difference concerns the outputs, because the models are designed to solve the overarching problem of finding relevant items and may get there through different means. In the case of interaction prediction, the prediction can be made by computing embeddings that need to be compared, by directly predicting scores for each items, or even by computing the likelihood of the items to be interacted with.

This framework focuses on embedding creation, thus

Designing a framework that fits any model is a challenging task why? different inputs: discussed in Section 3.2.1 could expect different shape of solutions e.g. embeddings, scores, probability distribution How we circumvented that? standardize outputs by limiting to pure embeddings let models free for training loop.

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.

$$\text{dot\_product\_score}(\boldsymbol{e}^{\text{user}}, \boldsymbol{e}^{\text{item}}) = \frac{\boldsymbol{e}^{\text{user}}}{\|\boldsymbol{e}^{\text{user}}\|} \cdot \frac{\boldsymbol{e}^{\text{item}}}{\|\boldsymbol{e}^{\text{item}}\|} \tag{5}$$

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.

$$\text{L2\_score}(\boldsymbol{e}^{\text{user}}, \boldsymbol{e}^{\text{item}}) = \sqrt[k]{\|\boldsymbol{e}^{\text{user}} - \boldsymbol{e}^{\text{item}}\|^k} \tag{6}$$

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:

$$L_{\text{reg}} = \boldsymbol{U}\boldsymbol{U}^T + \boldsymbol{I}\boldsymbol{I}^T \tag{7}$$

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:

$$\cos\left(\frac{2\pi}{\Delta}t\right), \sin\left(\frac{2\pi}{\Delta}t\right) \tag{8}$$

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. 5) 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 3.2 illustrates the architecture 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.

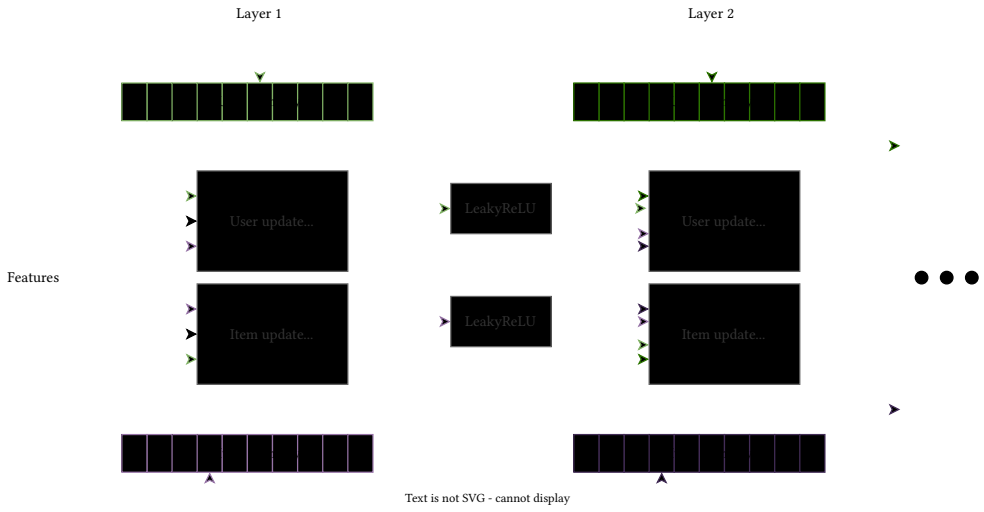

Figure 3.2: Architecture of LiMNet with two layers.

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 [6], 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.

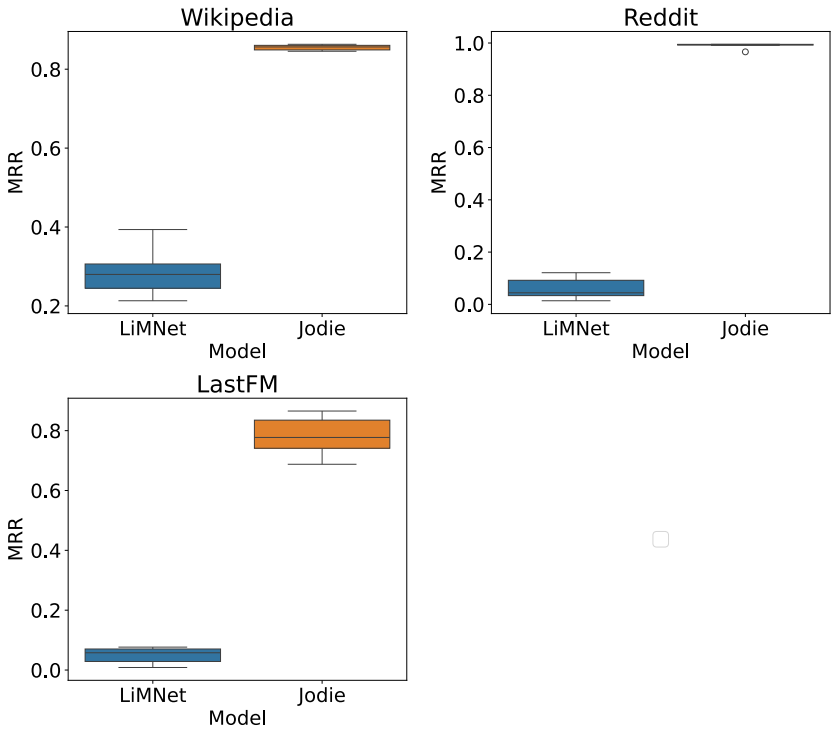

Figure 4.1: Comparison of LiMNet and Jodie models.

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:

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

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. 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. 5. In this experiment, we tried to systematically normalize the embeddings after each interaction; this

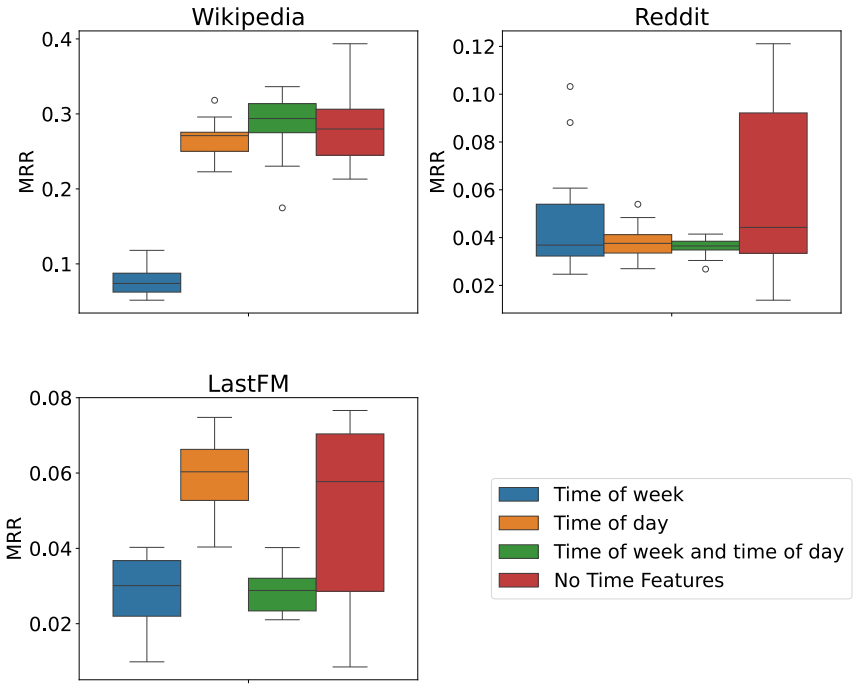

Figure 4.2: Performances of the LiMNet model with time features added.

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 3.2 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

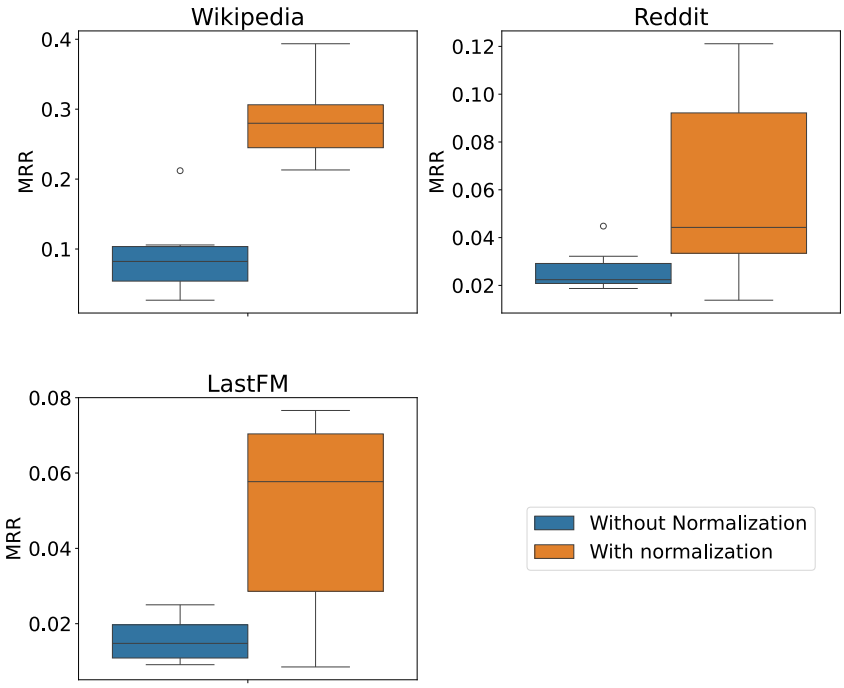

Figure 4.3: Performances of the LiMNet model with and without normalization of the embeddings.

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

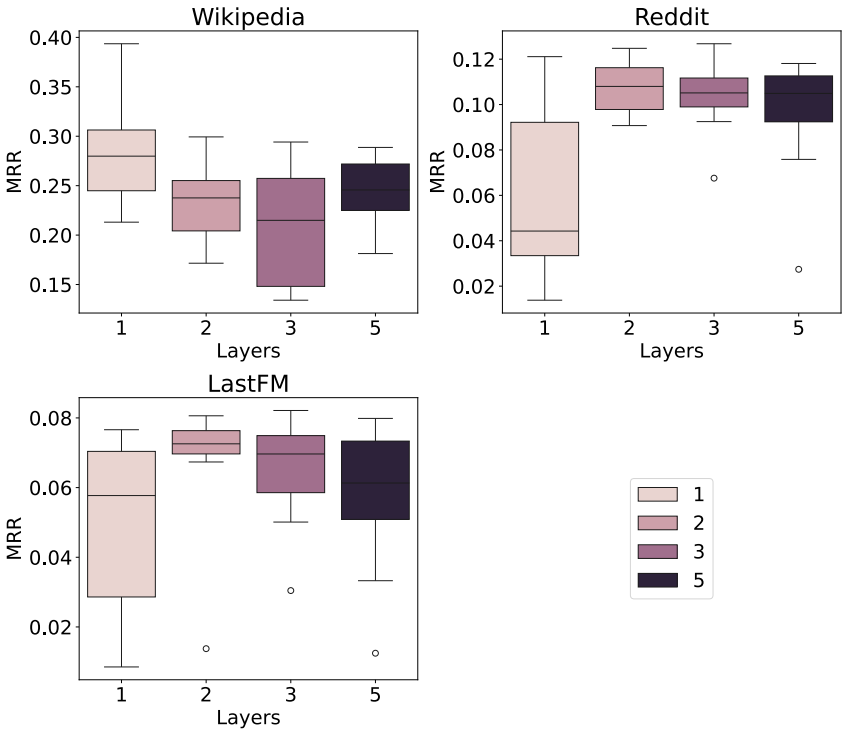

Figure 4.4: Performances of the LiMNet model with various numbers of stacked layers.

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

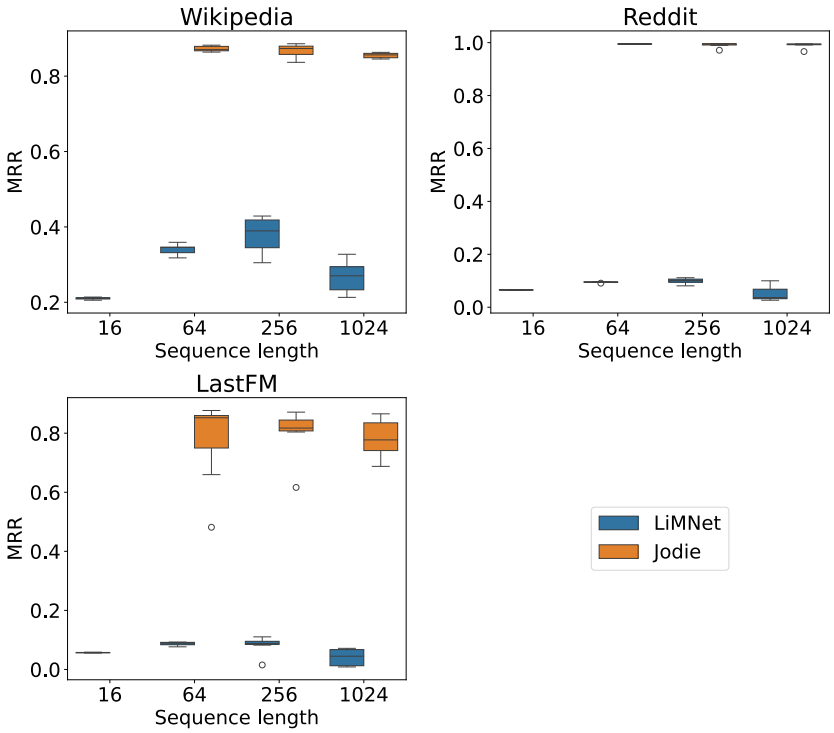

Figure 4.5: Effect of the sequence length on the models' performances.

one presented in the paper is the batching algorithm; in this project, interactions are processed in chronological order, while in [6] 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. 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 [9]. 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 [10].

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] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling," 2014. [Online]. Available: https://arxiv.org/abs/1412.3555

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

[10] 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.

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