



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 21, 2025

Supervisors: Šarūnas Girdzijauskas and Lodovico Giarretta

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 includefile-name 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 reproduce a 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

1. Introduction	1
1.1. Purpose/Motivation	1
1.2. Problem	2
1.3. Goals	2
1.4. Delimitation	2
1.5. Contribution	2
1.6. Ethics and Sustainability	2
2. Background	3
2.1. User-Item Link Prediction	3
2.2. Graph Representation Learning	4
2.3. Dynamic Graphs	5
2.4. Cross-RNN	6
2.5. LiMNet	8
3. Method	9
3.1. Datasets	9
3.2. Experimental framework	9
3.2.1. Preparation of the data	10
3.2.2. Batching Strategy	10
3.2.3. Training and evaluation loops	10
3.2.4. Comparison of the embeddings	10
3.2.5. Code maintainability	10
3.3. Adaptations to the Lightweight Memory Networks (LiMNet) architecture	11
3.3.1. Addition of time features	11
3.3.2. Normalization of the embeddings	11
3.3.3. Stacking several LiMNet layers	11
3.4. Exploration of the baselines	12
4. Results	13
4.1. Effects of the proposed improvements for LiMNet	14
4.2. Comparison with Jodie	14
4.3. Effect of the batching strategy	14
References	15

Acronyms and Abbreviations

CTDG – Continuous-Time Dynamic Graph: Dynamic graph where each edge is associated with a timestamp. 5, 6

GNN – Graph Neural Networks: Specialized neural networks that are designed for tasks whose inputs are graphs. 5

GRL – Graph Representation Learning: The task of computing high level representation of graphs, subgraphs, nodes or edges. Used as inputs for Machine Learning algorithms. 4, 5

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. 7

LiMNet – Lightweight Memory Networks: A neural network architecture that focuses on efficient memory utilization and lightweight design for temporal interaction network embeddings computations. vii, 2, 8, 9, 10, 11, 13, 14

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. 7

NLP – Natural Language Processing: Field of Machine Learning aiming at automatically interpreting spoken and written languages. 5

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. 6, 7

Temporal Interaction Network: Dynamic network where each interaction constitutes a punctual edge, i.e. an edge with no duration. 6

Chapter 1

Introduction

1.1. Purpose/Motivation

The rapid expansion of digital technology has resulted in the production of an overwhelming abundance of information, to the point that it is a challenge to find relevant and meaningful material among the multitude. To not only alleviate but also leverage this information overload, the interest have surged for search engines and recommendation systems. These two subjects 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: their search history, demographics, pasts interactions with the system, etc. to filter the information to display. Content personalization is the whole 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 output an item from a catalog. In order to measure the performance of such an algorithm, we need to know what items would be relevant for each users. 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 given as inputs to predict future interactions, that will themselves be added used for later inference. What sets this task apart from other self-supervised tasks is the relational aspect of the information used, each interaction explicitly connecting information with other interactions and the interacting elements. In addition, interaction order and temporality usually matters.

LiMNet is a simple Machine Learning model of a new kind that is designed to process interactions in a causal way, leveraging both relational information and the interactions order. This model proved its performance at solving the task of botnet detection in IoT network, 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 driving research question for this work is the following:

“Can LiMNet perform well for interaction prediction?”

1.3. Goals

1.4. Delimitation

1.5. Contribution

1.6. Ethics and Sustainability

Chapter 2

Background

This chapter provide 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 to a node. An item can be any kind of information the user is interested in. It could be web pages, music tracks, items in an e-commerce catalog, and so on... 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 item is the most relevant for a given user, which is the same as predicting which interaction should be added next in the graph.

A classical approach to that problem is to measure how close each item is to the user in the graph. 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.

In addition of the relationship between users and items, most real-world system provide rich information about the nature of each interactions, users, and items. For example, in a music streaming service, an interaction can have a type (stream, like, playlist add, ...), as well as a listening duration. While each song can have information attached about its genres, its length, and for users, their age, and their location. All of this information is typically processed by Machine Learning systems that provide reliable results

for numerical features. The challenge is then to meld traditional Machine Learning approaches for numerical features with graph-based methods for relational information.

Lichtenwalter et al. proposed to approach link prediction as a supervised Machine Learning problem, instead of scoring each edge, they try, given an edge, to predict if it will exist in the future. To include the relational data to their model, they add some of the closeness metrics discussed earlier to the users and items features[1]. This typical machine learning setup leads to switch from a straightforward prediction setting to a feature engineering approach when it comes to graph-based data. Instead of looking for the desired property in the structure of the graph, this approach will try to summarize the structure into rich representation compatible with machine learning algorithm. The goal is not to proxy the desired property, but to create a rich representation of the graph data that can be used by a machine learning algorithm.

All the previously mentioned methods presents one main drawback: each time a user want 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 allow 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 general subject in data mining, it can be used to classify graph structures such as protein graphs, to capture information from a subgraph for example by creating subgraph representation from a knowledge graph to feed into a Large Language Model. More commonly, GRL tries 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.

Two main approaches have emerged to create these embeddings, the first one is inspired by Natural Language Processing (NLP) and the second one from Computer vision. The first approach creates random walks along the graph and assimilate each node to a token, and each random walk to a sentence or a sample of text. This way, any methods that produce word embeddings can also be used to create nodes embeddings. This method can even be used to create embeddings for paths or subgraphs.

The second approach is inspired by convolutional networks used in computer vision. The idea is that convolutional networks see images as graphs of pixels and successively apply transformation to these pixels based on their direct neighborhood. Such operations can be applied similarly to general graphs, the main challenge being to accept neighborhood of varying size. These architectures are called Graph Neural Networks (GNN) and have proven very effective for GRL and link-prediction[2].

2.3. Dynamic Graphs

Most of the information we get from networks is dynamic, especially for user-item interaction networks where each interaction is usually happening 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 to exploit causality relationship 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 like 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[3], Zheng et al. explain two ways temporal information are commonly included into graph data. The first one considers a series of snapshot 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 to represent reality of a lot of system in a completely faithful way. 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 nodes embeddings is to maintain a memory of the embeddings and update them as interactions are read. One of the building block for this approach is DeepCoevolve[4], a model for link prediction that uses 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 notables performance upgrade.

JODIE[5] 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 time. operation carried over by the intensity function in DeepCoevolve.

DeePRed[6] is an other 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

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

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

Where t stands for the time step of the input \mathbf{i}_t . $\mathbf{o}(\mathbf{i}_t)$ marks the output of the layer and \mathbf{h}_t represent the memory of the layer after receiving the input \mathbf{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 iterate 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 than LSTM, in spite of its reduced cost.

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

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

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

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

$$\mathbf{h}_t^v = \mathbf{h}_{t-1}^v \quad \forall v \in \mathbb{U} \setminus \{u\} \cup \mathbb{I} \setminus \{i\} \quad (5)$$

Where u is the user interacting with item i at time t with feature \mathbf{f} , and g^u and g^i are tunable functions, comparable to the function g in Eq. 2. LSTM and GRU cells 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 do not require to pass 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[7], 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. For the purpose of this work, we will however 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 or deletions. If a new node is added to the graph, it’s embedding can be computed immediately, without a need to retrain the model. Node deletions are even easier to handle, all it takes is to delete the corresponding embedding from the memory. In practice, LiMNet has already proven it’s potential on the task of IoT botnet detection[7] and fraud detection[8], it is thus expected that it could yield satisfying results for link-prediction, while requiring less resources than State of the Art solutions.

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

The three datasets we use in this project are directly taken from the Stanford Large Network Dataset Collection (accessible at snap.stanford.edu/data/index.html). They have all been used in Kefato et al. in [5] and have been reused a large number of time since then because of their use of access and

- we use Wikipedia, reddit, lastfm
- describe Datasets
- reddit is a bigger Wikipedia
- lastfm is on the next level of bigness

3.2. Experimental framework

Evaluating embedding methods is challenging because it requires to accommodate for a wide variety of inputs and outputs 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 they can be approached as graphs, as time series, etc. In this section, we will precise the design decisions that led us to our final implementation, available at: ...

These decisions are grouped in 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 dataset 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 model tested only use the identifiers and the implicit order of the interaction. Whenever the time information is relevant to a model, the timestamp is used to compute custom features that are passed to the model in addition to the ids.

3.2.2. Batching Strategy

As pointed out by previous work processing 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 [5], and Kefato et al. removed the recursions from their model DeePRed by approximating the dynamic embeddings with static ones [6]. Inspired by the original LiMNet proposition [7], we decided instead to slice the data into sequences of fixed size with the argument that increasing the We use fixed sequence length and process sequences in parallel, in batches of sequences.

3.2.3. Training and evaluation loops

We realized that there is no unified way of processing the internals of each method. However, in order to conserve consistency, we make sure that each epoch is processed the same way. Each model is responsible for its own training over a given batch of sequences. Delivering the batches is managed by a common code.

3.2.4. Comparison of the embeddings

challenge: not all models are intended to exploit the embeddings the same way. example: l2 norm vs dot-product (cosine similarity) what we do: require models to produce embeddings for all items and for the user (or predicted item) to compare with. The scoring is done using both l2 and dot product, the better score is kept scoring and ranking is done in a unified way.

3.2.5. Code maintainability

Reusing existing code in research is not very common as progress are usually encountered by using a different viewpoint that requires a different approach. However, it is also important to reproduce experiments. This code

was written with the goal of being reusable, first by being as flexible as the scope of the project allowed it to be. Second though a thorough documentation targeted at helping future adaptations or re-implementations. This also goes with a functional approach, pure functions being conceptually easier to understand than complex objects with distributed state. Note however, that models required a more object-oriented approach to match PyTorch design.

3.3. Adaptations to the LiMNet architecture

The first goal of this work was to test how the LiMNet model would perform on a link-prediction task. The implementation we used of LiMNet is notably stripped down of its inputs and outputs maps, as well as the response layer used in the original paper to fit the specific needs of the task of IoT botnet detection [7].

In addition of this simplification, we tried to enhance the model with three separate additions: adding time features, normalizing the embeddings and stacking several LiMNet layers.

3.3.1. Addition of time features

We made the assumption that user behaviors follow cyclic patterns. Thus we tested if adding time features could improve the model. We did not have access to explicit timestamps, so we went for a simple approach. We computed time features as sine and cosine of the timestamp, with periods of a day and a week.

3.3.2. Normalization of the embeddings

This second improvement was inspired by the use of cosine similarity to evaluate the similarity between user and item's embeddings. cosine similarity only care about the angle between the embeddings. normalizing the embeddings to the unit sphere thus appeared as an efficient way not to converge to 0 while also focusing the model on meaningful transformations.

3.3.3. Stacking several LiMNet layers

Lastly we wanted to see whether LiMNet could be turned into a deep neural network. Thus we tried the following architecture: **Schema needed**. we used leaky relu as activation layers Normalization is applied at every step.

3.4. Exploration of the baselines

We tried 2 baselines to compare with: static embeddings, and Jodie We also tried DeePRed but couldn't reproduce the results.

Static embeddings consist of learning a static representation for each node. Used as a simple baseline, but is oblivious to the temporal relationships. Hard to apply to a real world system: need to be re-trained to account for each new interaction.

Jodie is conceptually close to LiMNet. Has x major differences:

- Mixes dynamic embeddings with one-hot embeddings
- Uses a projection mechanism to account for time elapsed between interactions
- Uses a dedicated loss function

Our implementation is truthful to the paper except for the batching algorithm. Jodie allows to dynamically account for new interactions in the network without re-training. Jodie is not designed for node insertions/deletions.

Chapter 4

Results

- Jodie is sensitive to the embedding size
- Adding layers to LiMNet doesn't improve the performance for link prediction
- Adding time features does not improve the performance
- Jodie can perform much better (???)
- Normalizing results seems to increase performances for LiMNet and Embeddings

Claim	Wikipedia		Reddit		Lastfm	
Changing embedding size Jodie	∅	∅	✓	✓	✓	
Adding layers LiMNet	✓	✓	✓	✓	✓	✓
Adding time features LiMNet	✓	✓	✓	✓	✓	~
Normalizing results LiMNet	~	~	✓	✓	✓	✓

- with little to no information the model is performing somewhat good

Experiments to conduct:

- Each model at its best
- LiMNet with time features (none, both, day, week)
- LiMNet without normalization (with/without)
- LiMNet at several layers (1, 3, 5, 2)
- Jodie at several embedding size (32, 64, 16, 48, 128)
- Models with a small sequence length

In this chapter, we present the results of the **6** experiments performed. First, we discuss in Section 4.1 the measured performances of the proposed improvements on the LiMNet architecture. Then, Section 4.2 present the results yielded by our implementation of Jodie[5], and we discuss the differences we noticed with the initial publication. Lastly, Section 4.3 exhibits the impact of the batching strategy on the two models.

4.1. Effects of the proposed improvements for LiMNet

4.2. Comparison with Jodie

4.3. Effect of the batching strategy

References

- [1] 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](https://doi.org/10.1145/1835804.1835837).
- [2] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, “A Comprehensive Survey on Graph Neural Networks,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4–24, 2021, doi: [10.1109/TNNLS.2020.2978386](https://doi.org/10.1109/TNNLS.2020.2978386).
- [3] 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](https://doi.org/10.1007/s11704-024-3853-2).
- [4] H. Dai, Y. Wang, R. Trivedi, and L. Song, “Deep Coevolutionary Network: Embedding User and Item Features for Recommendation,” arXiv:1609.03675, arXiv, Feb. 2017. doi: [10.48550/arXiv.1609.03675](https://doi.org/10.48550/arXiv.1609.03675).
- [5] 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](https://doi.org/10.1145/3292500.3330895).
- [6] 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](https://doi.org/10.1145/3442381.3450020).
- [7] L. Giarretta, 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](https://doi.org/10.1007/978-3-030-88418-5_29).

- [8] 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>

TRITA – EECS-EX 2024:0000

Stockholm, Sweden 2025

www.kth.se