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: April 22, 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

Lorem ipsum dolor.

Keywords

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

Sammanfattning

Lorem ipsum dolor.

Nyckelord

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

Acknowledgments

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do.

Lorem.

Lorem ipsum dolor.

Stockholm, April 2025
Titouan Mazier

Contents

Acronyms and Abbreviations

CTDG – **Continuous-Time Dynamic Graph**: Dynamic graph were each edge is associated with a timestamp. 4

GNN – **Graph Neural Networks**: Specialized neural networks that are designed for tasks whose inputs are graphs. 3

GRL – **Graph Representation Learning**: The task of computing high level representation of graphs, subgraphs, nodes or edges. Used as inputs for Machine Learning algorithms. 3, 4

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. 5, 6

LiMNet – **Lightweight Memory Networks**: A neural network architecture that focuses on efficient memory utilization and lightweight design for temporal interaction network embeddings computations. 6

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. 5, 6

NLP – **Natural Language Processing**: Field of Machine Learning aiming at automatically interpreting spoken and written languages. 3

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. 5, 6

Temporal Interaction Network: Dynamic network were each interaction constitute a punctual edge, i.e. an edge with no duration. 4

Chapter 1
Background

This chapter provide the background for the project. In Section 1.1 we provide an overview of link prediction and the classical solutions for the problem, in Section 1.2 we further develop the concept of graph embedding and some common methods to create them. Then in Section 1.3 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 1.4. Finally we present the model of interest for this work in Section 1.5 and why we believe that it is a relevant addition to the task of link prediction.

1.1. Link Prediction

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 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.

1.2. 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].

1.3. 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 phenomenons 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 1.2 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 1.4) 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.

1.4. Cross-RNN

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

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

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

Where t stands for the time step of the input $\boldsymbol{i}_t$. $\boldsymbol{o}(\boldsymbol{i}_t)$ marks the output of the layer and $\boldsymbol{h}_t$ represent the memory of the layer after receiving the input $\boldsymbol{i}_t$. The functions f and g can vary depending on the nature of the RNN but they will rely on weights, tuned during the model training. Popular RNN architectures try to keep a memory of long-term knowledge. Typically, the Long-Short Term Memory (LSTM) architecture maintains two distinct memories, a short term one and a long term one. The Gated Recurrent Unit (GRU) architecture 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 $\boldsymbol{h}_t$, it maintain a memory for all nodes in the graph $\boldsymbol{H}_t = (\boldsymbol{h}_t^u)_{u \in \mathbb{U}} \cup (\boldsymbol{h}_t^i)_{i \in \mathbb{I}}$, where $\mathbb{U}$ and $\mathbb{I}$ are the sets of users and items in the graph. For each interaction $(u, i, t, \boldsymbol{f})$ the memory is updated as follow:

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

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

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

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

Where u is the user interacting with item i at time t with feature $\boldsymbol{f}$, and g^u and g^i are tunable functions, comparable to the function g in Eq. 2. LSTM and GRU cells 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 however is mostly an issue for the training of the model, because processing one input in inference do not require to pass through the entire sequence.

1.5. LiMNet

Lightweight Memory Networks (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.

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.

[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.

[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.

[4] 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.

[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.

[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.

[7] 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.

[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