Completing a Temporal Knowledge Base Graph on Diachronic Entity Embedding and Graph Representation Learning

Le Ngoc Lam
Department of Computer Science
Ha Noi University of Science and Technology
HA NOI, VN 100000
lam.ln232036m@sis.hust.edu.vn

Abstract

This report addresses the challenge of completing knowledge graphs by focusing on temporal knowledge graph completion methods, which account for changes in data over time. The report explores the combined model DE-SimplE with the graph convolution neural network to enhance performance in temporal environments. Experiments will be conducted on ICEWS14 and ICEWS05-15 datasets, using Hit@k and MRR metrics.

Keywords: Knowledge graph completion, temporal knowledge graph completion, chronological embedding

1 Introduction

Knowledge graphs store information about entities (such as people, places, or things) and their relationships. They are designed to help computers understand the meaning of data more easily by organizing it in a structured, meaningful way that can be queried and analyzed effortlessly.

Knowledge Graph Completion (KGC) is a task that involves predicting missing information or links in a knowledge graph. In this context, a knowledge graph stores structured information that represents relationships between entities such as people, places, and things. KGC techniques use machine learning algorithms to analyze existing data and infer new connections between entities. The goal of KGC is to fill the gaps in knowledge graphs, making them more complete and useful for various applications such as search engines, recommendation systems, and question-answering systems. By completing the knowledge graph, we can enhance our understanding of relationships between entities, making it easier to extract insights and make informed decisions from the data.

Temporal Knowledge Graph Completion (TKGC) is an extension of knowledge graph completion that incorporates time as a factor. TKGC involves predicting missing relationships or links in a temporal knowledge graph, a database representing relationships between entities over time. The task is to infer and predict the duration, frequency, and sequence of events occurring between different entities at various times. TKGC techniques use machine learning algorithms to analyze temporal patterns in the existing data and infer new time-based connections.

Many knowledge graph completion models perform well on static knowledge graphs but cannot capture temporal features in temporal knowledge graphs. For this reason, Rishab Goel et al., (1) proposed a diachronic entity embedding approach to enable static knowledge graph completion models to capture temporal characteristics. The model proposed by the authors to represent this approach is DE-SimplE. This model is based on the SimplE static knowledge graph embedding model (2) combined with diachronic entity embeddings, producing high temporal knowledge graph dataset results.

Preprint. Under review.

This report combines DE-SimplE with graph convolution to enhance structure information in temporal knowledge. The results significantly outperform the original model (DE-SimplE) on the metrics Hit@10, Hit@3, Hit@1, and MRR on the datasets ICEWS14. The report content will be presented in the following sections. Section 1 introduces graph knowledge, knowledge graph completion, temporal knowledge graph completion, and how to solve it with static/dynamic models in the context of this report. Section 2 summarizes some techniques for dealing with static knowledge graph completion, and the diachronic embedding model for temporal knowledge graph completion. Section 3 mentions the proposed combination of diachronic entity embedding and graph convolution networks for temporal knowledge graph completion. Section 4 presents hardware, and software with an overview of the datasets, related metrics, model setup, and also results obtained along with an evaluation and explanation. Section 5 will have some conclusions as well as future work.

2 Related Works

Knowledge Graph: is a structured representation of knowledge, consisting of nodes (entities) and edges (relationships) that connect them. For example, in a geographical knowledge graph, nodes might represent entities such as “Hai Ba Trung” “Hanoi,” and “Vietnam,” while edges represent relationships like “capital of” or “located in.” One of the main advantages of a knowledge graph is that it allows for more natural and intuitive data querying compared to traditional databases. Instead of relying on complex SQL queries, users can query a knowledge graph with natural language questions such as "Who is the president of the U.S.?" or "How many provinces are there in Vietnam?" For instance, Google uses a massive knowledge graph to enhance the effectiveness of its search engine. Similarly, companies like Amazon and Facebook use knowledge graphs to better understand customer preferences and behaviors, allowing them to provide more personalized recommendations and advertisements. According to Ke Liang et al (3), knowledge graphs are divided into three types:

- **Static Knowledge Graph**: This is the most common type of knowledge graph. A static knowledge graph, as in Figure 1, is typically represented as SKG = {V, R, F}, where, V is the set of entities or nodes in the graph, R is the set of relations or edges in the knowledge graph, $F = \{(e_h, r, e_t) \mid e_h, e_t \in V, r \in R\}$ is the set of triples, where e_h is the head entity, e_t is the tail entity, and r is the relationship between them.

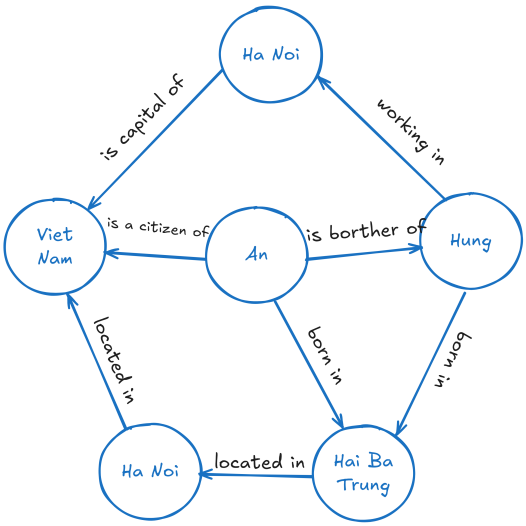

Figure 1: Sample of Static Knowledge Graph

- **Temporal Knowledge Graph**: This is a sequence of static knowledge graphs at different time points: $TKG = \{SKG_1, SKG_2, SKG_3, \ldots, SKG_t\}$, where $SKG_t = \{V, R, F_t\}$ and t represents different time points. In a temporal knowledge graph, as in Figure 2, each piece of knowledge is represented by a quadruple $F = \{(e_h, r, e_t, t)\}$. Similar to a static knowledge graph, a temporal knowledge graph consists of entities (nodes) and relationships

(edges), but now the entities can have different versions depending on their corresponding time points.

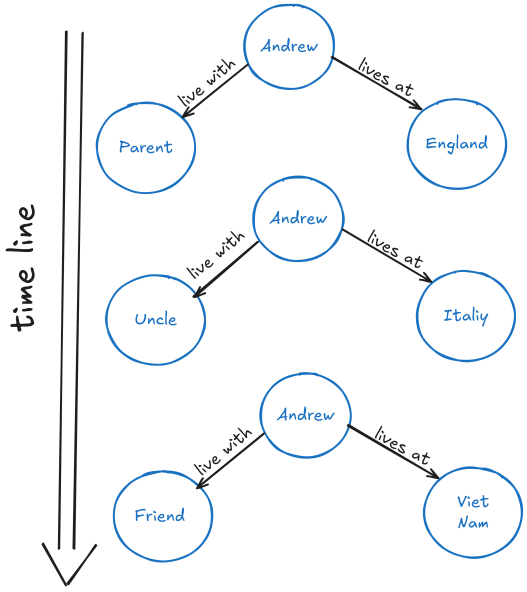

Figure 2: Sample of Temporal Knowledge Graph

- **Multimodal Knowledge Graph:** This type of knowledge graph incorporates non-textual data such as images and audio. Unlike static knowledge graphs, multimodal knowledge graphs include non-textual entities, which can be images, audio, or files.

Knowledge Graph Completion: Knowledge graphs are often incomplete, meaning they lack head entities, tail entities, or relationships due to the difficulty of manually collecting all possible connections between entities. Additionally, large knowledge graphs built automatically from internet data are prone to omissions and errors. This leads to poor performance in practical applications. Knowledge Graph Completion (KGC) addresses this issue by predicting missing links in the graph using existing data, often with machine learning techniques. The goal is to infer logical rules from existing knowledge to fill in missing relationships, such as:

$$(\text{Hanoi, is the capital of, Vietnam}) \wedge (\text{West Lake, is in, Hanoi}) \rightarrow (\text{West Lake, is in, Vietnam}).$$

By completing knowledge graphs, we can improve understanding of entity relationships, enhance search accuracy, make personalized recommendations, answer complex queries, and predict drug efficacy by analyzing gene, protein, and disease connections. KGC models calculate the likelihood of a queried relationship based on scoring functions, but temporal and multimodal knowledge graphs require additional reasoning due to the inclusion of temporal and visual information. This report will introduce methods for completing static and temporal knowledge graphs.

- **Static Knowledge Graph Completion:** For a static knowledge graph, the goal is to find the missing relations or entities in triples (e_h, r, e_t) from existing knowledge, with query types $(?, r, e_t)$, $(e_h, ?, e_t)$, and $(e_h, r, ?)$. According to Ke Liang et al., (4), there are three main approaches: embedding-based, path-based, and rule-based. Embedding-based models, like TransE, learn low-dimensional vector representations for entities and relations, predicting missing links by ranking triples. Other models such as TransH, TransR, and RESCAL use different techniques to represent entities and relationships. Path-based models infer relationships by analyzing paths between entities. For example, if two entities share a path of length k, they may have a relationship inferred from the other relationships along the path. Rule-based models, like AMIE, predict missing relationships by applying predefined or learned logical rules, offering better interpretability, especially in domains requiring transparency such as healthcare and finance.

- **Temporal Knowledge Graph Completion:** The challenge of temporal knowledge graph completion aims to infer missing data from a temporal knowledge graph, focusing on predicting head or tail entities, relations, or timestamps in a quadruple (e_h, r, e_t, t). Query types include $(?, r, e_t, t), (e_h, ?, e_t, t), (e_h, r, ?, t)$, and $(e_h, r, e_t, ?)$. Applications include predicting stock prices or forecasting disease risk. The challenge lies in handling the complexity of temporal data, as events occur at different intervals. Two main approaches exist (4): RNN-based models, which use recurrent neural networks (RNNs) like LSTM or GRU to capture sequential patterns, and non-RNN-based models, which extend static knowledge graph methods to temporal graphs. Tensor factorization methods, such as Tucker and Canonical Polyadic decomposition, represent temporal knowledge graphs as four-dimensional tensors, allowing the learning of entity and relation embeddings. Other models utilize attention mechanisms for historical context inference or time embeddings, such as TComplEx and HyTE.

This report focuses on studying and developing the method proposed by Rishab Goel et al. (1), which applies diachronic embedding combined with graph convolution networks to complete temporal knowledge graphs.

3 Methodology

This section of the methodology will be structured as follows section 3.1 will formulate the problem, and Section 3.2 mentions SimplE as the basic method to improve. Sections 3.3 and 3.4 will gradually improve SimplE by using Diachronic Embeddings and Graph Convolution Networks.

3.1 Problem Formulation

Knowledge Graphs (KG) often include time-sensitive events that show the relationships between entities over time. Many methods have been proposed to infer new events for a knowledge graph, and this challenge is referred to as knowledge graph completion. Several models have been introduced to address this challenge, with notable examples being embedding-based knowledge graph completion models such as RotatE (5), TransE (3), PairE (6), ... Although knowledge graph embedding techniques have been proven successful in knowledge graph completion, they are primarily designed for static knowledge graphs. An increasing challenge is the creation of temporal knowledge graph embedding models. Specifically, the input and output requirements for this problem are as follows:

The input is a knowledge graph $G = \{V, R, T, F\}$ where:

- V is the set of entities, or nodes, in the graph.
- R is the set of relations, or links, in the knowledge graph.
- T is the set of timestamps.
- $F = \{(e_h, r, e_t, t) \mid e_h, e_t \in V, r \in R, t \in T\}$ is the set of quadruples (head entity, relation, tail entity, timestamp).

The output:

- Let W be the set of all possible links in the knowledge graph ($G \subset W$). The problem requires to find W from G.

3.2 SimplE

SimplE (Simple Embedding Model for Link Prediction) is a simple but effective method for embedding entities and relationships in knowledge graphs. This method is introduced to perform the problem of link prediction, which means predicting missing relationships between entities in a structured knowledge system.

How the SimplE works:

To answer the question "How many possibilities are there that the triple <h,r,t> occurs?" We can answer that by computing the score of it. So first of all we have to embed entities and relations to the same space. Each of entities head and tail will be embedded into two different vectors:

- z_h Embedding vector when the entity is head
- z_t Embedding vector when the entity is tail

Relations r also embed to 2 vectors:

- $z_{forward}$ Embedding vector when the relation is forward
- $z_{inverse}$ Embedding vector when the relation is inverse

Example of forward and inverse relation: the triple of forward relation forward <An, read, book>, also exists inverse relation <book, is read by, An>.

The score of the tripe <h,r,t> will be computed by this equation:

$$score(h, r, t) = \frac{1}{2}(z_h.z_{forward}.z_t + z'_h.z_{inverse}.z'_t) \tag{1}$$

The multiply operator of entities and the relation in the score function is similar to cosine similarity.

3.3 Diachronic Embeddings combined with the SimplE method

Section 3.2 introduces the SimplE method, to improve embedding entities in SimpleE we use Diachronic Embedding.

The diachronic entity embedding, DEEMB: $(V, T) \rightarrow \Psi$, is a mapping function that maps every pair (v, t), where $v \in V$ and $t \in T$, to a set of diachronic entity embedding vectors Ψ.

Static knowledge graph completion models can be transformed into temporal knowledge graph completion models by replacing their static entity embedding vectors with diachronic entity embedding vectors. Let $z_v^t \in \mathbb{R}^d$ be a vector in Ψ (i.e., DEEMB$(V, T) = (\dots, z_v^t, \dots)$). The vector z_v^t is computed as follows:

$$z_v^t[n] = \begin{cases} a_v[n]\sigma(\omega_v[n]t + b_v[n]), & \text{if } 1 \leq n \leq \gamma d, \\ a_v[n], & \text{if } \gamma d < n < d. \end{cases} \tag{2}$$

where $a_v \in \mathbb{R}^d$, and $\omega_v, b_v \in \mathbb{R}^{\gamma d}$ are parameter vectors, and σ is the activation function.

According to the authors, entities can have features that change over time and others that remain constant. The first γd dimensions of the vector in Equation 2 capture the time-dependent features (dynamic embedding), while the remaining $(1-\gamma)d$ dimensions capture the time-independent features (static embedding). The hyperparameter γ controls the proportion of dynamic dimensions relative to the output embedding vector, where $0 \leq \gamma \leq 1$.

We observe that the equation for diachronic entity embedding consists of two parts: $a_v[n]\sigma(\omega_v[n]t + b_v[n])$ is the dynamic embedding part, and $a_v[n]$ is the static embedding part. Although the formula could be simplified to:

$$z_v^t[n] = a_v[n]\sigma(\omega_v[n]t + b_v[n]),$$

In Equation 2, the parameters ω_v and b_v help the model adjust the entity embedding vector a_v based on the time t. The authors use the sine function for the activation function σ, as the sine function can capture various recurring temporal patterns.

According to Equation 2, diachronic entity embeddings are divided into two parts:

- **Static Embedding** ($a_v[n]$): Each entity in the static knowledge graph is mapped to an embedding vector. This vector is learned during the training process.
- **Dynamic Embedding** ($a_v[n]\sigma(\omega_v[n]t + b_v[n])$): The embedding vector is scaled with time through the activation function σ.

The dynamic embedding accounts for γ, while the static entity embedding accounts for $(1 - \gamma)$ (as illustrated in Figure 6), where γ is a hyperparameter and $0 \leq \gamma \leq 1$.

Figure 3: The diachronic embedding vector consists of two components: dynamic embedding (accounting for a proportion of γ) and static entity embedding (accounting for a proportion of $1-\gamma$)

The dynamic embedding is calculated as:

$$H = a_v \sigma(\omega_v t + b_v) \tag{3}$$

where a_v, ω_v, and b_v are learnable embedding vectors through the training process, with γd dimensions for each entity v, t is time (in days), and σ is the activation function (sine). Assuming the embedding vector has a single dimension, we have the graph depicting the variation of H over time t.

As we can observe, when time changes, the dynamic embedding of entities changes according to the sine function. ω_v and b_v determine how a characteristic changes over time, and a_v determines the maximum value of that characteristic (i.e., the more significant the characteristic, the higher the value of a_v). Rishab Goel et al., (1) utilized the sine function as the activation function, arguing that the sine function captures periodic events due to its infinite repetition.

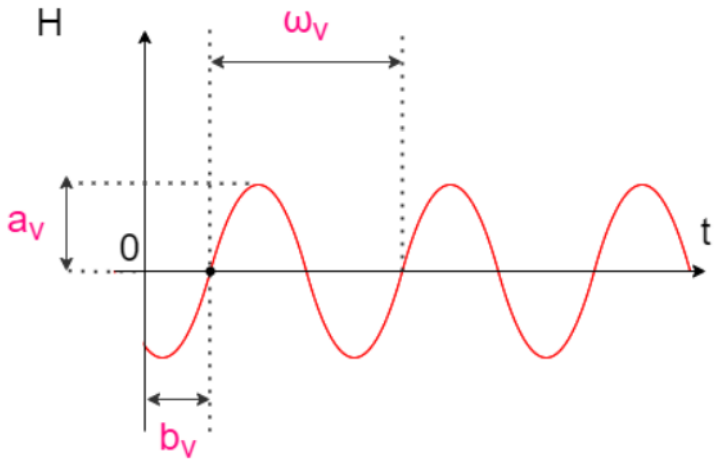

Figure 4: The variation of the dynamic embedding H over time t with the sine activation function sine

From Figure 4, we can observe that if the maximum time (t) becomes too large (e.g., 365 days or 3650 days), consecutive days will have almost the same value, making it challenging for the model to learn the patterns of events occurring in short intervals (e.g., 1 to 2 days). Therefore, in the provided source code by the authors, the dynamic embedding is calculated according to the formula:

$$H = a_v^d \sin(\omega_v^d t_d + b_v^d) + a_v^m \sin(\omega_v^m t_m + b_v^m) + a_v^y \sin(\omega_v^y t_y + b_v^y) \tag{4}$$

where t_d, t_m, and t_y represent the day, month, and year of the event, respectively. a_v^d, a_v^m, and a_v^y are the entity embeddings for day, month, and year, respectively. Similarly, ω_v^d, ω_v^m, ω_v^y and b_v^d, b_v^m, b_v^y are the corresponding vectors. By separating the day, month, and year, the maximum value of t decreases, making it easier for the model to learn the patterns of events occurring in short periods. For convenience, we simplify Equation 4 as follows:

$$H = a_v^{d,m,y} \sin(\omega_v^{d,m,y} t_d + b_v^{d,m,y}) \tag{5}$$

3.4 Diachronic Embeddings combined with the SimplE method and Graph Convolutional Networks

3.4.1 Graph Convolution Neural Networks

Graph Convolutional Networks (GCNs) are a type of convolutional neural network specifically designed to work with graph-structured data. GCNs learn to represent the vertices in the model by taking the information of neighbors.

$$G^{l+1} = \sigma(\tilde{A}G^l W^l) \tag{6}$$

Each vertice in the graph at layer l can be represented by the equation 6. In this equation:

- $G^{(l)}$: matrix to represent the vertices of layer l.
- $W^{(l)}$: weight matrix for layer l l.
- $\tilde{A}$: This matrix gets information on self-vertices and their neighbors. It computes by $\tilde{A} = adjacencymatrix + identitymatrix$. $Adjmatrix$ for relationship and I for self-vertices.
- σ: Activation function

After L layers, we get the matrix to present the graph as G^L. We call G_v^L is represented of vertice v_{th}.

3.4.2 Diachronic Embeddings combined with Graph Convolution Networks

The equation 2 in section 3.3 can be rewritten by using the graph convolution network as follows:

$$z_n^t[n] = \begin{cases} a_v[n] \cdot \sigma(\omega_v[n] \cdot t + b_v[n]) & \text{if } 1 \leq n \leq \gamma_1 d \\ a_v[n] & \text{if } \gamma_1 d \leq n \leq \gamma_2 d \\ G_v[n] & \text{if } \gamma_2 d \leq n \leq d \end{cases}$$

Now the entity will take information on the structure of the Knowledge Graph, and information about time. The presence of an entity can be more clear. We call this method is Diachronic Embedding-SimplE-Graph Convolution Networks (DE-SimplE-GCNs)

4 Experiments

This section presents the hardware and software used during the experiments. Next, the report will discuss the datasets utilized. Finally, it presents the experimental results of the proposed model and compares them with other models.

4.1 Hardware and Software Configuration

Hardware: The experiments were conducted entirely on Kaggle websites as in Table 1.

Table 1: Hardware used during the experiments

Description	**Kaggle**
Source	`https://www.kaggle.com/`
GPU	Nvidia P100
GPU Memory	16GB
GPU Processing Speed	1.32GHz
RAM	32GB
Storage Capacity	19.5GB

Software: This report uses the Python programming language and the Pytorch framework.

4.2 Datasets

ICEWS, which stands for *Integrated Crisis Early Warning System*, is a database containing political events with specific timestamps. From this, several temporal knowledge graphs were derived by García-Durán et al. Two knowledge graphs used in the experimental section include ICEWS14 and ICEWS05-15. ICEWS14 contains events that occurred in 2014 with timestamps in days, while ICEWS05-15 includes events from 2005 to 2015.

The overview of datasets used for experiments can be seen in Table 2.

Table 2: Overview of the two datasets

	ICEWS14	**ICEWS05-15**
Entities ($\mathcal{V}$)	7,128	10,448
Relations ($\mathcal{R}$)	230	251
Timestamps ($\mathcal{T}$)	365	4,017
Training Set ($\mathcal{F}_{train}$)	72,826	386,962
Validation Set ($\mathcal{F}_{valid}$)	8,941	46,275
Test Set ($\mathcal{F}_{test}$)	8,963	46,092
Total ($\mathcal{G}$)	90,730	479,329

4.3 Metrics

For each quadruple $f = (v, r, u, t) \in F_{\text{test}}$, generate two queries: $(v, r, ?, t)$ and $(?, r, u, t)$. For each query, the model will rank all remaining entities in $u \cup \bar{C}_{f,u}$, where $\bar{C}_{f,u} = \{u' : u' \in V, (v, r, u', t) \notin G\}$. Let r be the rank of the correct entity predicted by the model.

Mean Reciprocal Rank (MRR):
The mean of all reciprocal ranks for both head and tail entities in all test cases is the MRR of the model. The more effective the model, the higher the MRR. The following equation can be used to determine MRR:

$$\text{MRR} = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{r_i} \tag{8}$$

where n is the total number of predicted entities.

Hit@k:
Hit@k is a metric commonly used to evaluate recommendation systems, search engines, and knowledge graph completion models. It measures the proportion of correct results appearing in the top k positions of the ranked list of entities. If the model finds 5 correct entities within the top k predicted entities, the Hit@k is $5/k$.

$$\text{Hit@k} = \frac{1}{n} \sum_{i=1}^{n} 1[r_i \leq k] \tag{9}$$

4.4 Settings

Table 3: Hyperparameters Used in the Experiments

Hyperparameter	**Value**
Learning Rate	0.001
Batch Size	512(ICEW14), 1024(ICEW05-15)
Epochs	50
Negative Ratio	150
Dropout Rate	0.4
γ_1	0.36
γ_2	0.36
d	100
Number layers of GCN	2

4.5 Results

Compare metrics Diachronic Embedding-SimplE-Graph Convolution Networks are very strong because take information about the structure of the Graph and information about time, so let's see the result of this method with other methods:

Table 4: Performacne of DE-SimplE-GCN with other methods

Model	**ICEW14**				**ICEWS05-15**			
	Hit@1	**Hit@3**	**Hit@10**	**MRR**	**Hit@10**	**Hit@3**	**Hit@1**	**MRR**
DE TransE	7.4	40.4	65.3	27.6	6.9	38.7	63.6	26.7
DE-SimplE	33.1	51	**67.5**	44.8	**33.1**	**50.7**	**68.5**	**45**
DE-SimplE- GCN	**33.6**	**56.9**	67.3	**45**	28.4	43.7	61.2	39.2

We can see that in small data(ICEW14 dataset), the performance of our proposal is better than Diachronic Embedding TransE and Diachronic Embedding SimpleE. But on the large scale (ICEW05-15 dataset) it becomes worse depending on 3 reasons:

- Large graphs can make information blurry. The entity has to receive a lot of information from neighbors so it does not know important information.
- The entity can only take information from K-hop around it, maybe the important information can be located far away.
- Over-smoothing, when the representations of different nodes become too similar after a few information propagations.

Besides this, the loss function of the DE-SimplE-GCN method can converge faster than other methods.

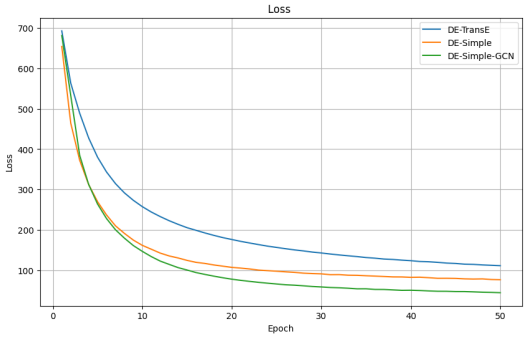

Figure 5: Loss evaluation of 3 methods

5 Conclusion and Future Works

Conclusion: This report has focused on introducing knowledge bases, knowledge graphs, and temporal knowledge graphs, along with tasks and models for completing static knowledge graphs and temporal knowledge graphs, respectively.

This proposes the Diachronic Embedding-SimplE with Graph Convolution (MDE-SimplE-GCN) model for completing temporal knowledge graphs. Experimental results on the ICEWS14 dataset demonstrate performance improvements over the original DE-SimplE model.

Future Works:

1. Use some sequence machine learning models combined with graphs to improve performance.

2. This report still lacks experiments on the change of hyperparameters. So maybe the performance in the section 4.5 is not the best. In the future need more experiments about it.

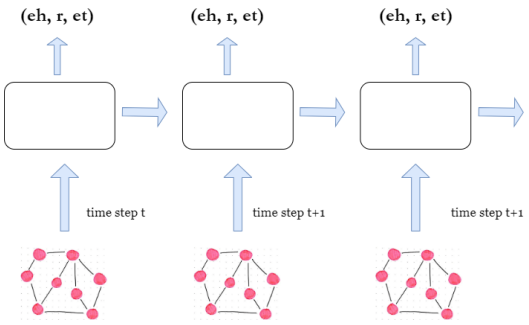

Figure 6: Propose when using LSTM

References

[1] R. Goel, S. M. Kazemi, M. Brubaker, and P. Poupart, "Diachronic embedding for temporal knowledge graph completion," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, pp. 3988–3995, 2020.

[2] S. M. Kazemi and D. Poole, "Simple embedding for link prediction in knowledge graphs," *Advances in neural information processing systems*, vol. 31, 2018.

[3] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko, "Translating embeddings for modeling multi-relational data," *Advances in neural information processing systems*, vol. 26, 2013.

[4] K. Liang, L. Meng, M. Liu, Y. Liu, W. Tu, S. Wang, S. Zhou, X. Liu, and F. Sun, "A survey of knowledge graph reasoning on graph types: Static," *Dynamic, and Multimodal*, 2022.

[5] Z. Sun, Z.-H. Deng, J.-Y. Nie, and J. Tang, "Rotate: Knowledge graph embedding by relational rotation in complex space," *arXiv preprint arXiv:1902.10197*, 2019.

[6] L. Chao, J. He, T. Wang, and W. Chu, "Pairre: Knowledge graph embeddings via paired relation vectors," *arXiv preprint arXiv:2011.03798*, 2020.