Recurrent Event Network for Reasoning over Temporal Knowledge Graphs

Woojeong Jin[1] **He Jiang**[1] **Changlin Zhang**[1] **Pedro Szekely**[2] **Xiang Ren**[1,2]
[1]Department of Computer Science, University of Southern California
[2]Information Sciences Institute, University of Southern California
{woojeong.jin, jian567, changlin.zhang, xiangren}@usc.edu
pszekely@isi.edu

Abstract

Recently, there has been a surge of interest in learning representation of graph-structured data that are dynamically evolving. However, current dynamic graph learning methods lack a principled way in modeling *temporal*, *multi-relational*, and *concurrent* interactions between nodes—a limitation that is especially problematic for the task of temporal knowledge graph reasoning, where the goal is to predict unseen entity relationships (i.e., events) over time. Here we present Recurrent Event Network (RE-NET)—a novel neural architecture for modeling complex event sequences—which consists of a *recurrent event encoder* and a *neighborhood aggregator*. The event encoder employs an RNN to capture (subject, relation) or (object, relation)-specific patterns from historical, multi-relational interactions between entities. The neighborhood aggregator summarizes concurrent, multi-hop entity interactions within each time stamp. An output layer is designed for predicting forthcoming events. Extensive experiments[1] on temporal link prediction over four public TKG datasets demonstrate the effectiveness and strength of RE-NET, especially on multi-step inference over future time stamps.

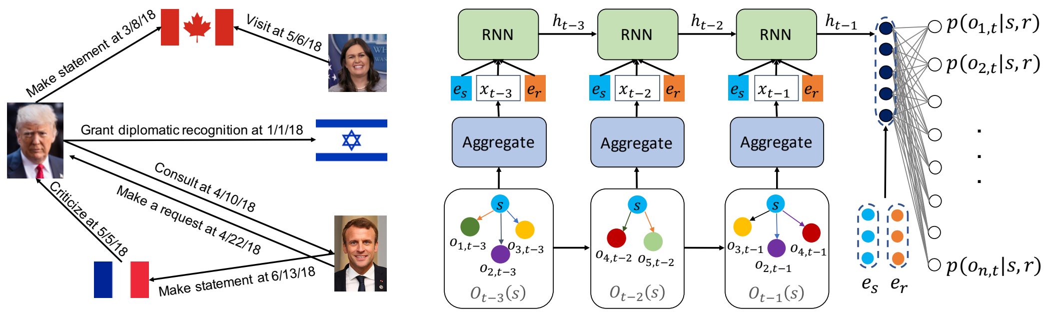
1 Introduction

Representation learning on dynamically evolving graph-structured data has emerged as an important machine learning task in a wide range of applications, such as social network analysis [16], question answering [25, 26], and event forecasting [22, 23]. This task becomes particularly challenging when dealing with multi-relational graphs where complex interaction patterns exist between nodes—*e.g.*, in reasoning over temporal knowledge graphs (TKGs). Recent studies on TKG reasoning focus on embedding time expressions into low-dimensional space [9, 6, 13]. However, these methods either lack a principled way to model *temporal dependencies* between *multi-relational* entity interactions, or ignore the consolidation of *concurrent interactions* within the same time stamp [22]. Furthermore, extending existing dynamic network embedding models [8, 29, 16, 8] to perform TKG reasoning is non-trivial—they do not capture multi-relational entity interaction patterns and require expensive re-training when doing multi-step prediction over future time stamps.

In this paper, we propose a novel neural architecture for modeling temporal, multi-relational, graph-structured data, called Recurrent Event Network (RE-NET). Key ideas of RE-NET are based on the following observations: (1) a TKG can be viewed as a sequence of multi-relational interactions between entities; (2) multiple interactions may occur within the same time stamp; (3) strong dependencies exist between temporally adjacent interactions; (4) multi-relational, local neighborhood is helpful to predict future interactions. To address the limitations of prior works, RE-NET introduces

[1]Code and data are published at https://github.com/INK-USC/RE-Net.

Preprint. Work in progress.



(a) An example of temporal KG

(b) Overview of the RE-NET architecture

Figure 1: **Illustration of (a) temporal knowledge graph and (b) the Recurrent Event Network architecture.** RE-NET employs an RNN to capture (s, r)-specific interactions with object entities $O_t(s)$ (modeled by a neighborhood aggregator) at different time t, and adopts the hidden state of RNN h_t and (subject, relation) embeddings (e_s, e_r) for predicting forthcoming entity interactions.

an *event sequence encoder* and a *neighborhood aggregation module*. The event sequence encoder captures temporal and multi-relational dynamics by encoding the past interactions between entities (i.e., events) with a recurrent neural network. The neighborhood aggregation module resolves concurrent interactions at the same time stamp by consolidating neighborhood information. Also, we propose various neighborhood aggregation modules and a multi-relational aggregator demonstrates its effectiveness among them. A classification layer is designed to predict unseen entity relationships for the current time stamp, given prior encoder state, subject entity, and relation. We adopt multi-class cross entropy loss to learn the RE-NET model and perform multi-step inference for predicting forthcoming events on temporal knowledge graphs over time.

We evaluate our proposed method on temporal link prediction task, by testing the performance of multi-step inference over time on four public temporal knowledge graph datasets. Experimental results demonstrate that RE-NET outperforms state-of-the-art models of both static and temporal graph reasoning, showing its better capacity to model temporal, multi-relational graph data with concurrent events. We further show that RE-NET can perform effective multi-step inference to predict unseen entity relationships in a distant future.

2 Related Work

Our work is related to representation learning methods for static, multi-relational graphs, previous studies on temporal knowledge graph reasoning, and recent advancements in recurrent graph models.

Temporal KG Reasoning and Link Prediction. There are some recent attempts on incorporating temporal information in modeling dynamic knowledge graphs. [22] presented Know-Evolve which models the occurrence of a fact as a temporal point process. However, this method is built on a problematic formulation when dealing with concurrent events, as shown in Section F. Several embedding-based methods have been proposed [9, 13, 6] to model time information. They basically embed the associate into a low dimensional space such as relation embeddings with RNN on the text of time [9], time embeddings [13], and temporal hyperplanes [13]. However, these models do not capture temporal dependency and cannot generalize to unseen time stamps because their models only handle the current time information.

Static KG Completion and Embedding. Extensive studies have been done on modeling static, multi-relation graph data for link prediction. There are methods which embed entities and relations into low-dimensional spaces [3, 27, 24, 7]. Among these methods, Relational Graph Convolutional Networks (RGCN) [20] generalized the previous GCN works [12] by dealing with directed, multi-relational graphs such as knowledge graphs. These methods achieve high accuracy on reasoning on static knowledge graphs. However, they cannot deal with temporal evolution on knowledge graphs.

Recurrent Graph Neural Models. There have been some studies on recurrent graph neural models for sequential or temporal graph-structured data [19, 2, 17, 21, 18]. These methods adopt a message passing framework for aggregating nodes' neighborhood information (*e.g.*, via graph convolutional operations). GN [19, 2] and RRN [17] update node representations by a message passing scheme between time stamps. EvolveGCN [18] and GCRN [21] introduce a RNN to update node representations across different time stamps. In contrast, our proposed method, RE-NET, augments a RNN with message passing procedure between entity neighborhood to encode temporal dependency between (concurrent) events (i.e., entity interactions), instead of using the RNN to memorize historical information about the node representations.

3 Proposed Method: RE-NET

3.1 Temporal Knowledge Graph Reasoning

A *temporal knowledge graph* (TKG) is a multi-relational, directed graph with time-stamped edges (relationships) between nodes (entities). An *event* E_t is defined as a time-stamped edge, *i.e.*, (subject entity, relation, object entity, time) and is denoted by a quadruple (s, r, o, t). A TKG is built upon *a sequence of event quadruples* ordered ascending based on their time stamps, *i.e.*, $\mathcal{E} = \{E_{t_i}\}_i = \{(s_i, r_i, o_i, t_i)\}_i$ (with $t_i < t_j, \forall i < j$), where each time-stamped edge has a direction pointing from the subject entity to the object entity.[2] The task of **reasoning over TKG** aims to predict unseen relationships with object entities given $(s, r, ?, t)$ (or subject entities given $(?, r, o, t)$), based on the observed events in the TKG. To model lasting events which span over a time range, *i.e.*, $(s, r, o, [t_1, t_2])$, we simply partition such event into a sequence of time-stamp events $\{E_{t_1}, ..., E_{t_2}\}$. We leave more sophisticated modeling of lasting events as future work.

3.2 Recurrent Event Network

We propose RE-NET to capture the temporal dynamics for predicting forthcoming events and to summarize the concurrent events within the same time stamp. RE-NET consists of a Recurrent Neural Network (RNN) as an event sequence encoder and a neighborhood aggregation module to capture the information of local graph structures. In the rest of this section, we will formalize the sequential event prediction task and provide details on the two proposed modules in RE-NET.

Sequential Event Prediction in TKG. The key idea in RE-NET is to view the TKG, denoted by $\mathcal{E}$, as *a sequence of event sets* $\{\mathcal{E}_1, ..., \mathcal{E}_T\}$, where $\mathcal{E}_t = \{(s, r, o, t') \in \mathcal{E} | t' = t\}$ denote the set of events occurring at time stamp t, and capture their temporal, multi-relational dependency by constructing a sequential event prediction model. Specifically, we aim to model the joint probability of the TKG or event sequence $\mathcal{E}$, *i.e.*,$\mathbb{P}(\mathcal{E}) = \prod_{t=1}^{T} \mathbb{P}(\mathcal{E}_t | \mathcal{E}_{t-1}, \dots, \mathcal{E}_1)$. By imposing assumptions on (1) Markov property of event sets $\{\mathcal{E}_t\}$ and (2) conditional independence between concurrent events, we can rewrite the joint probability $\mathbb{P}(\mathcal{E})$ into the following form (cf. Appendix A for detailed derivation):

$$\mathbb{P}(\mathcal{E}) = \prod_{t=1}^{T} \prod_{E_t \in \mathcal{E}_t} \mathbb{P}(o_t | s_t, r_t, \{\mathcal{E}_{t-1}, \dots, \mathcal{E}_{t-m-1}\}) \cdot \mathbb{P}(s_t, r_t | \mathcal{E}_{t-1}, \dots, \mathcal{E}_{t-m-1}). \tag{1}$$

In this work, we consider the case that probability of (s_t, r_t) is independent of the past m event sets $\{\mathcal{E}_{t-1}, \dots, \mathcal{E}_{t-m-1}\}$, and model it using uniform distribution, leading to $\mathbb{P}(E_t | \mathcal{E}_{t-1}, \dots, \mathcal{E}_{t-m-1}) \propto \mathbb{P}(o_t | s_t, r_t, \{\mathcal{E}_{t-1}, \dots, \mathcal{E}_{t-m-1}\})$. Furthermore, we assume that $\mathbb{P}(o_t | s_t, r_t, \{\mathcal{E}_{t-1}, \dots, \mathcal{E}_{t-m-1}\})$ depends only on events that are related to s_t, and focus on modeling the following probability:

$$\mathbb{P}(\mathcal{E}) \propto \prod_{t=1}^{T} \prod_{E_t \in \mathcal{E}_t} \mathbb{P}(o_t | s, r, \{O_{t-1}(s), ..., O_{t-m-1}(s)\}), \tag{2}$$

where $\mathcal{E}_{t_i}$ becomes $O_{t_i}(s)$ which is a set of object entities interacted with subject entity s under *all* relations at time stamp t_i. We ignore the time index of (s, r) for simplicity in the rest of the paper.

Event Sequence Encoder. We further parameterize the conditional probability of object entities (given a subject-relation pair (s, r)[3] and its past interactions) using a function f, defined as follows:

$$\mathbb{P}(o_t | s, r, \{O_{t-1}(s), ..., O_{t-m-1}(s)\}) = f(e_s, e_r, h_{t-1}(s, r)), \tag{3}$$

[2] The same triple (s, r, o) may occur multiple times in different time stamps, yielding different event quadruples.

[3] Probability conditioned on an object-relation pair (o, r) can be defined in a similar way.

where $O_{t_i}(s)$ is a set of entities interacted with s at t_i. $e_s, e_r \in \mathbb{R}^d$ are learnable embedding vectors specified for subject entity s and relation r, and $h_{t-1}(s,r)$ is a history vector which encodes the information from the m object sets (s,r) interacted in the past, *i.e.*, $\{O_{t-1}(s), ..., O_{t-m-1}(s)\}$. In our implementation, f is a multilayer perceptron with one hidden layer followed by a softmax layer to output class (entity) probability.

We assume that the next set of objects can be predicted with a previous object history. To track the history of interactions, we introduce an event sequence encoder based on RNN as follows

$$h_t(s,r) = \text{RNN}(e_s, e_r, g(O_t(s)), h_{t-1}(s,r)). \tag{4}$$

In each time step, besides the history $h_{t-1}(s,r)$, we add the aggregation of neighbor representation $g(O_t(s))$ We also use a subject embedding and a relation embedding as well as aggregation of objects as the input of RNN to make the RNN subject-relation specific. If we don't include the subject and relation embeddings, then the history will be equivalent to all subject-relation pairs.

Neighborhood Aggregation. A subject entity can make interactions with multiple objects under relation r or all relations at the same time stamp. To encode the entity neighborhood information to a fixed-length input for our RNN encoder, we define an aggregation module $g(\{e_o\})$ to collect information from relation-specific neighbors $\{e_o : o \in O_t^r(s)\}$ (cf. Sec. 3.3), or multi-relational neighbors $\{e_o : o \in O_t(s)\}$ (cf. Sec. 3.4).

3.3 Basic Neighborhood Aggregator Architectures

Here we discuss different choices for the aggregate function $g(\cdot)$, which capture different kinds of neighborhood information for each subject entity and relation, *i.e.*, (s,r). We limit $O_t(s)$ to neighbors under relation r, denoted by $O_t^r(s)$. We assume that the next set of objects can be predicted with the previous object history under the same relation.

Here, we propose several basic aggregation operations for summarizing the neighborhood information, given a subject entity s and a relation r. More complex aggregation will be introduced in Sec. 3.4.

Mean Aggregator. The baseline method is to simply take the element-wise mean of the vectors in $\{e_o : o \in O_t^r(s)\}$, where $O_t^r(s)$ is a set of objects interacted with s under r at t. But the mean aggregator treats all neighboring objects equally, and thus ignores the different importance of each neighbor entity.

Attentive Aggregator. We define an attentive aggregator based on the additive attention introduced in [1] to distinguish the important entities for (s,r). The aggregate function is defined as $g(O_t^r(s)) = \sum_{o \in O_t^r(s)} \alpha_o e_o$, where $\alpha_o = \text{softmax}(v^\top \tanh(W(e_s; e_r; e_o)))$. $v \in \mathbb{R}^d$ and $W \in \mathbb{R}^{d \times 3d}$ are trainable weight matrices. By adding the attention function of the subject and the relation, the weight can determine how relevant each object entity is to the subject and relation.

Pooling Aggregator. Based on the mean aggregator, we design trainable and non-linear pooling aggregator. This pooling aggregator is equivalent to the convolutional operator used in [12]. This aggregator takes the form as $g(O_t^r(s)) = \sigma\Big(W \cdot \frac{1}{|O_t^r(s)|} \sum_{o \in O_t^r(s)} e_o\Big)$, where $W \in \mathbb{R}^d$ is a trainable weight matrix. Different from the mean aggregator, the pooling aggregator uses another weight matrix W and a nonlinear activation function (*e.g.* ReLU).

3.4 Multi-relational Graph (RGCN) Aggregator

We introduce a multi-relational graph aggregator based on [20] to aggregate representations of neighbors with different relations. This is a general aggregator that can incorporate information from: (1) multi-relational neighbors; (2) multi-hop neighbors; and (3) neighbors of a cumulative graph which is built from $\{\mathcal{E}_t, ..., \mathcal{E}_{t-q}\}$. This aggregator collects representations of multi-hop neighbors by multiple layers of the neural network. The number of layers determines the depth to which the node reaches to aggregate information from its local neighborhood. Also, information from the cumulative neighbors $\{O_t(s), ..., O_{t-q}(s)\}$ can be gathered from the cumulative graph. Its aggregate function is defined as:

$$g(O_t(s)) = h_s^{(l+1)} = \sigma\Big(\sum_{t'=t-q}^{t} \sum_{r \in R} \sum_{o \in O_{t'}^r(s)} \frac{1}{c_s} W_r^{(l)} h_o^{(l)} + W_0^{(l)} h_s^{(l)}\Big), \tag{5}$$

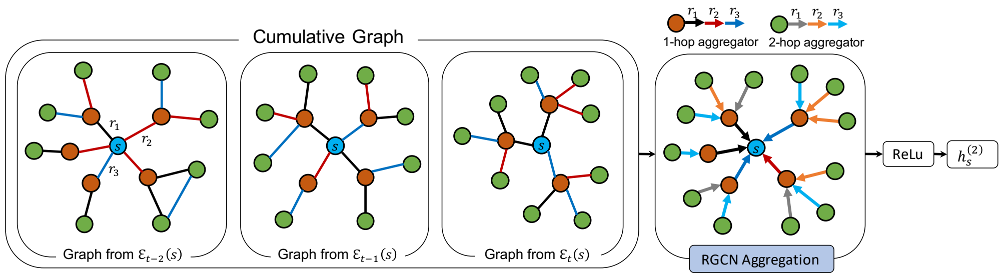


Figure 2: **Illustration of the multi-relational graph (RGCN) aggregator.** The blue node corresponds to node s, red nodes are 1-hop neighbors, and green nodes are 2-hop neighbors. Different colored edges are different relations. In this figure, we build a cumulative graph from $\{\mathcal{E}_t, \mathcal{E}_{t-1}, \mathcal{E}_{t-2}\}$ and we get $h_s^{(2)}$ from a two-layered RGCN aggregator on the graph.

where c_s denotes the number of cumulative neighbors of node s from time $t-q$ to t, and $h_o^{(l)}$ is the hidden state of node o in the l-th layer of the neural network. We define the activation function $\sigma(\cdot)$ as ReLu. The multi-relational graph aggregator accumulates transformed feature vectors of neighboring nodes. To differentiate the node representation at layer $l+1$ and node representation at layer l, we use a self-loop weight matrix $W_0^{(l)}$. This aggregator is depicted in Fig. 2.

The major issue of this aggregator is that the number of parameters grows linearly with the number of relations. In practice, this can easily lead to overfitting on rare relations and models of very large size. Thus, we adopt the block-diagonal decomposition [20], where each relation-specific weight matrix is decomposed into a block-diagonal and low-dimensional matrix. $W_r^{(l)}$ in (5) is defined as $diag(A_{1r}^{(l)}, ..., A_{Br}^{(l)})$ where $A_{kr}^{(l)} \in \mathbb{R}^{(d^{(l+1)}/B)\times(d^{(l)}/B)}$ and B is the number of basis matrices. The block decomposition reduces the number of parameters for relation-specific interaction matrix $W_r^{(l)}$ and it can alleviate overfitting on rare relations by the basis weight matrices.

3.5 Inference and Parameter Learning of RE-NET

Multi-step Inference over Time. At inference time, given (s, r), RE-NET performs multi-step inference to predict forthcoming events, *i.e.*, $\{O_i^r(s)\}_{i=t_0}^{t_0+\Delta t}$. For example, reasoning for c time steps from last time stamp t yields entities $\{O_{t+1}^r(s), O_{t+2}^r(s), \ldots, O_{t+c}^r(s)\}$. During multi-step inference, the encoder state is updated based on current predictions, and will be used for making next predictions. That is, for each time step we rank the candidate entities and select top-k entities as current predictions. We maintain the history as a sliding window of length p, so the oldest interaction set will be detached and new predicted entity set will be added to the history.

Parameter Learning via Entity Prediction. The (object) entity prediction can be viewed as a multi-class classification task, where each class corresponds to one object entity. To learn weights and representations for entities and relations, we adopt a multi-class cross entropy loss to the model's output.The loss function for the predicted o_t is defined as:

$$\mathcal{L} = -\sum_{(s,r,o,t)\in\mathcal{E}} \sum_{c=1}^{M} y_c \log(p(o = c|s, r)), \tag{6}$$

where $\mathcal{E}$ is set of events, and y_c is a binary indicator (0 or 1) of whether class label c is the correct classification for prediction o. $p(o = c|s, r)$ is the probability that o is in class c. We use the softmax function on (3) to get the probability. We summarize the details on learning RE-NET in Algorithm 1.

Computational Complexity Analysis. Here we analyze the time complexity of Algorithm 1. Assume that the degree of each entity is d, and we have L layers of aggregation, the time complexity of each aggregation operation is $O(d^L)$. As we unroll for m time steps and the back-propagation through time has linear time complexity $w.r.t.$ m, the time complexity for each epoch is $O(|E|md^L)$.

Algorithm 1: Learning Parameters of RE-NET

Input: Events $\mathcal{E} = \{E_i\}_i = \{(s_i, r_i, o_i, t_i)\}_i$
Output: A trained classifier $f : (s, r; \{O_{t-1}(s), ..., O_{t-m-1}(s)\}) \rightarrow p(o_{i,t}|s, r)$

while *Parameters in* RE-NET *does not change* **do**
    foreach (s_i, r_i, o_i, t_i) *in* E **do**
        Get history $\{O_{t-1}(s_i), ..., O_{t-m-1}(s_i)\}$
        set $h(s_i, r_i) \leftarrow 0$
        for $i = t - m - 1$ *to* $t - 1$ **do**
            Aggregate neighborhood representations $x_i \leftarrow g(O_i(s))$
            Update history vector $h(s_i, r_i) \leftarrow \text{RNN}(e_{s_i}, e_{r_i}, x_i, h(s_i, r_i))$
        Compute $\hat{y} = f(e_{s_i}, e_{r_i}, h(s_i, r_i))$
        $\hat{y} \leftarrow softmax(\hat{y})$
        Compute cross entropy loss based on (6).
    Update model parameters.

4 Experiments

We evaluate our proposed method on three benchmark tasks: (1) predicting future events on two event-based datasets; (2) predicting future facts on two knowledge graphs which include facts with time spans, and (3) studying parameter sensitivity of our proposed method. Section 4.1 summarizes the datasets, and the supplementary material contains additional information. In all these experiments, we perform predictions on time stamps that are not seen during training.

4.1 Experimental Set-up

Datasets. We use four datasets: 1) two event-based temporal knowledge graphs and 2) two knowledge graphs where temporally associated facts have meta-facts as $(s, r, o, [t_s, t_e])$ where t_s is the starting time point and t_e is the ending time point. The first group of graphs includes Integrated Crisis Early Warning System (ICEWS18) [4] and Global Database of Events, Language, and Tone (GDELT) [14]. The second group of graphs includes WIKI [13] and YAGO [15].

Evaluation Setting and Metrics. For each dataset, we split it into three subsets, i.e., train(80%)/valid(10%)/test(10%), by time stamps. Thus, (times of train) < (times of valid) < (times of test). Note that this is different from the previous way which randomly picks train, valid, and test sets regardless of time stamps. We report Mean Reciprocal Ranks (MRR) and Hits@1/3/10, using the filtered version of the datasets as described in [3].

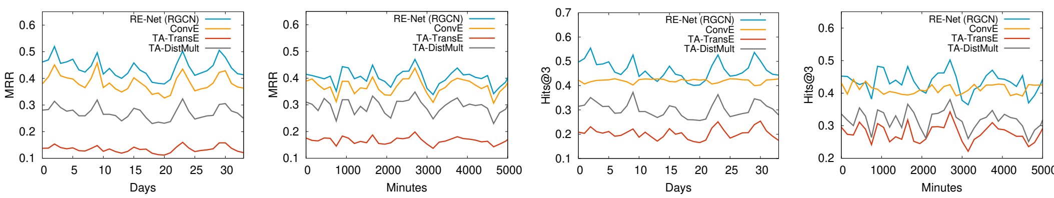
Model details for RE-NET. We use Gated Recurrent Units [5] as our event sequence encoder, where the length of history is set as $p = 10$. We use a multi-layer perceptron with one hidden layer followed by a softmax layer for f in (3). At inference time, RE-NET performs multi-step prediction across the time stamps in dev and test sets. In each time step we save top-10 entities to use them as concurrent past history for next prediction. We set the size of entity/relation embeddings to be 200 and the learning rate to be 0.001. The model is trained by the Adam optimizer. We set the weight decay rate to 0.00001. All experiments were done on GeForce GTX 1080 Ti. We use two-layer RGCN in the RGCN aggregator with block dimension 2×2.

Baseline Methods. We compare our approach to baselines for static graphs and temporal graphs: **(1)** ***Static Methods.*** By ignoring the edge time stamps, we construct a static, cumulative graph for all the training events, and apply multi-relational graph representation learning methods including TransE [3], DisMult [27], ComplEx [24], R-GCN [20], and ConvE [7]. **(2)** ***Temporal Reasoning Methods.*** We also compare with state-of-the-art temporal reasoning method for knowledge graphs, including Know-Evolve[4] [22], TA-TransE/DistMult [9], HyTE [6], and TTransE [13]. Experiment settings and implementation details of baseline methods are described in Section C.

[4]*: We found a problematic formulation in Know-Evolve when dealing with concurrent events (Eq. (3) in its paper) and a flaw in its evaluation code. The performance dramatically drops after fixing the evaluation code. Details of this issues are discussed in Section F.

Table 1: Performance comparison on temporal link prediction (average metrics in %) over two event-based TKG datasets. RE-NET with RGCN aggregator achieves the best results.

	Method	ICEWS18 - filtered				GDELT - filtered			
		MRR	Hits@1	Hits@3	Hits@10	MRR	Hits@1	Hits@3	Hits@10
Static	TransE [3]	17.56	2.48	26.95	43.87	16.05	0.00	26.10	42.29
	DisMult [27]	22.16	12.13	26.00	42.18	18.71	11.59	20.05	32.55
	ComplEx [24]	30.09	21.88	34.15	45.96	22.77	15.77	24.05	36.33
	R-GCN [20]	23.19	16.36	25.34	36.48	23.31	17.24	24.94	34.36
	ConvE [7]	37.67	29.91	40.80	51.69	36.99	28.05	40.32	50.44
Temporal	Know-Evolve* [22]	3.27	3.23	3.23	3.26	2.43	2.33	2.35	2.41
	HyTE [6]	7.31	3.10	7.50	14.95	6.37	0.00	6.72	18.63
	TTransE [13]	8.36	1.94	8.71	21.93	5.52	0.47	5.01	15.27
	TA-TransE [9]	12.85	0.00	19.04	37.53	16.62	0.00	27.65	42.53
	TA-DistMult [9]	28.53	20.30	31.57	44.96	29.35	22.11	31.56	41.39
	RE-NET (mean)	42.38	35.80	44.99	54.90	39.15	30.84	43.07	53.48
	RE-NET (attn)	41.46	34.67	44.19	54.44	38.07	29.44	42.26	52.93
	RE-NET (pool)	41.35	34.54	44.05	54.35	37.99	30.05	41.40	52.18
	RE-NET (RGCN)	**43.20**	**36.63**	**45.58**	**55.91**	**40.21**	**32.54**	**43.53**	**53.83**



(a) ICEWS18 (MRR) (b) GDELT (MRR) (c) ICEWS18 (H@3) (d) GDELT (H@3)

Figure 3: Performance of temporal link prediction over future time stamps. We report filtered MRR and Hits@3 (average metrics in %) on the test sets of ICEWS18 and GDELT datasets.

4.2 Performance Comparison on Temporal Knowledge Graphs.

In this section we compare our proposed method with other baselines. RE-NET (mean), RE-NET (attn), RE-NET (pool), and RE-NET (RGCN) denote our method with mean, attentive, pooling, and multi-relational graph aggregators, respectively. The test results are obtained by averaged metrics over the entire test sets on datasets.

Performance on Event-based TKGs. Table 1 summarizes results on two event-based datasets: ICEWS18 and GDELT. Our proposed method outperforms all other baselines on these datasets. Among all variants of aggregation functions, the RGCN aggregator shows the best performances while the mean, attentive and pooling aggregator show little lower performances. Interestingly, temporal baselines do not show better performances than static methods. This is because their methods try to model the current time information regardless of the past interactions. For example, TA-TransE designed time-aware representations for relation embeddings which only see the current time stamp, and thus it cannot generalize to unseen time stamps. However, RE-NET captures the temporal dependencies between events and can predict future entities even at the unseen time stamps.

Performance on Public KGs. Table 2 shows the performances on two knowledge graphs: WIKI and YAGO. RE-NET (attn) outperforms all other baselines on the WIKI dataset, whereas RE-NET (RGCN) shows the best performances on the YAGO dataset. Performances of temporal baselines show different aspects compared to the event-based datasets; TA-TransE and TA-DistMult show decent performances which is comparable to our proposed method. This is due to the characteristics of the datasets. WIKI and YAGO datasets has facts that are valid within a time span. These facts can range in a long time and even in train and test sets, and they do not occur multiple times. This is advantageous to methods that do not see past interactions. For example, TA-TransE shows improvement over TransE on WIKI and YAGO datasets, but it does not on the ICEWS18 dataset.

Table 2: Performance comparison on temporal link prediction (average metrics in %) on two public temporal knowledge graphs, i.e., WikiData and YAGO.

	Method	WIKI - filtered				YAGO - filtered			
		MRR	Hits@1	Hits@3	Hits@10	MRR	Hits@1	Hits@3	Hits@10
Static	TransE [3]	19.30	14.01	24.01	25.82	34.78	20.27	47.23	53.35
	DisMult [27]	26.15	25.66	26.30	27.06	54.58	53.00	55.19	57.24
	ComplEx [24]	50.84	50.47	51.08	51.39	64.47	63.45	65.11	65.74
	R-GCN [20]	37.57	34.79	39.66	41.90	41.30	35.08	44.44	52.68
	ConvE [7]	49.57	48.23	51.07	52.53	64.32	63.79	65.97	67.60
Temporal	Know-Evolve* [22]	00.09	00.01	00.02	00.10	00.07	0	0	00.04
	HyTE [6]	29.82	21.67	33.86	43.67	31.26	15.58	43.91	51.94
	TTransE [13]	33.51	23.94	41.54	47.96	33.35	15.21	48.95	58.22
	TA-TransE [9]	49.92	48.34	51.25	52.04	58.56	50.21	**66.32**	68.27
	TA-DistMult [9]	48.09	46.01	49.51	51.70	62.72	59.30	64.94	68.19
	RE-NET (mean)	48.30	45.86	49.36	53.03	65.51	63.85	66.06	68.03
	RE-NET (attn)	**51.72**	**50.60**	**52.12**	**53.72**	**65.79**	64.50	66.00	67.82
	RE-NET (pool)	45.15	41.41	46.98	52.57	63.65	61.25	64.76	67.45
	RE-NET (RGCN)	50.47	49.80	52.03	53.16	65.69	**64.83**	**66.32**	**68.48**

4.3 Performance of Prediction over Time.

Figs. 3 shows the performance comparisons over different time stamps on the ICEWS18 and GDELT datasets with filtered metrics. RE-NET outperforms other baselines with the MRR metric on both datasets. However, RE-NET and ConvE are comparable on the Hits@3 metric. We notice that with the increase of time step, the performance of RE-NET is getting lower and the difference between RE-NET and ConvE is getting smaller as shown in Fig. 3b. This is because RE-NET predicts events based on history, but the history is updated with predicted entities at inference. We also find that performance gaps on the GDELT are smaller than ICEWS18. GDELT has finer granularity of time than ICEWS18, and thus it requires a longer history for prediction.

4.4 Comparison between Different Neighborhood Aggregators.

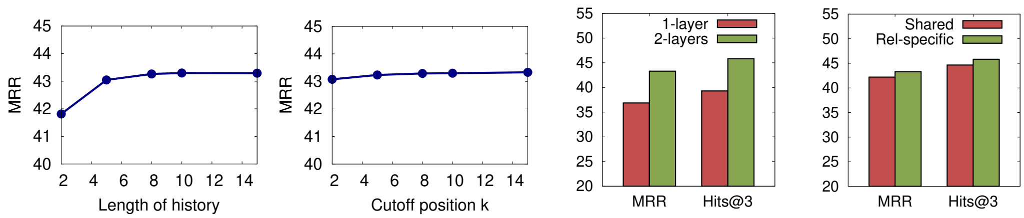
We observe different behavior of each aggregator depending on a dataset. We first notice that RE-NET (pool) show the worst performances among variants of aggregators. Pooling aggregator is built on the mean aggregator with additional parameter and non-linear activation function. We suppose that the performance is due to overfitting which is caused by additional parameters by observing performance loss of RE-NET (pool) from RE-NET (mean) on all datasets. On the other hand, RGCN aggregator outperforms other aggregators on the most datasets. This aggregator has the advantage of seeing a wide range of neighbors not limited to neighbors under the same relation. Also, the aggregator has relation-specific weights to distinguish neighbors of important relations, and it shows effectiveness which we will see in Section 4.5.

4.5 Study of Parameter Sensitivity

We study the effect of model hyper-parameters in RE-NET, including the length of RNN history, cutoff position k to save entities as concurrent history in inference. Furthermore, we study the hyper-parameters of RGCN aggregator. We report the performance change of RE-NET (RGCN) on the ICEWS18 dataset by varying the hyper-parameters.

Length of Past History in Event Sequence Encoder. The event sequence encoder takes the sequence of past interactions up to p previous histories. We examine the effectiveness of various length of past histories. Figure 4a shows the performances with varying length of past histories. When RE-NET uses longer histories, MRR is getting higher. However, the MRR is not likely to go higher when the length of history is 5 and over. This implies that long history does not make big differences.

Cut-off Position k at Inference Time. To do multi-step prediction, RE-NET should save current prediction as concurrent recent history for the next prediction. We cut off top-k entities on ranking results as concurrent events. Fig. 4b shows the performances with choosing different cutoff position



(a) Length of past history (b) Cutoff position k (c) # layers of RGCN (d) Weights in RGCN

Figure 4: Performance study on model hyper-parameters. We study the effects of (a) length of RNN history in event sequence encoder, (b) cutoff position used in entity prediction at inference time, (c) number of RGCN layers in neigborhood aggregation, and (d) weight sharing schema used by RGCN.

k. When k is larger, the performance is marginally higher. We notice that RE-NET is not sensitive to the cutoff position, which gives robustness to predict future entities.

Hyper-parameters of RGCN Aggregator. The RGCN aggregator has parameters to examine: number of layers of RGCN and relation-specific weights in RGCN. Fig. 4c shows the performances according to different numbers of layers of RGCN. We notice that 2-layered RGCN improves the performances considerably compared to 1-layered RGCN since 2-layered RGCN aggregates more information. Also, RGCN aggregator adopts relation-specific weights to distinguish the messages through different relations. We compare the relation-specific weights to the shared weight across all relations. Weight sharing in 1-layer RGCN aggregator is similar to the pooling aggregator. Using relation-specific weight improves the results as shown in Fig. 4d.

5 Theoretical Analysis

Here we analyze the model capacity of RE-NET of capturing complex time-invariant local structure like [10], as well as the emerging global community structure as [28]. Due to the space constraint, we state the main results here and leave the detailed discussion to Section G.

Theorem 1 *Let $\{\mathcal{E}_i\}_{i=1}^{\tau}$ be the snapshot of temporal knowledge graph after τ time-steps. Let $h_v^0 \in \mathbb{R}^d, v \in \{s_i\} \cup \{o_i\}$ to be the input feature representation for Algorithm 1 of each entity node v. Suppose that there exists a fixed positive constant $C \in \mathbb{R}^+$ such that $||h_v^0 - h_{v'}^0|| > C$ for all pair of all pair of entities v, v'. Then we have that $\forall \epsilon > 0$, there exist a parameter setting Θ for* RE-NET *s.t. after $K = 4$ layers of aggregation,*

$$|h_{v,\tau}^K - c_{v,\tau},| < \epsilon, \forall v \in \mathcal{V}, \forall \tau \in [T],$$

where $h_{v,\tau}^K$ are output values generated by RE-NET *and $c_{v,\tau}$ are clustering coefficients of $\{\mathcal{E}_i\}_{i=1}^{\tau}$.*

Observation 1 *Consider a temporal graph under stochastic block model described in Section G.2. Let $h_v^0 \in \mathbb{R}^d, v \in \{s_i\} \cup \{o_i\}$ to be the input feature representation for Algorithm 1 of each node. Suppose that a constant portion p_c of input representations can be linearly separated by a hyperplane, while the representation of other nodes lies on the hyperplane. There exists a parameter setting of* RE-NET *that can output the probability that new node j connected to node i.*

6 Conclusion

In this work, we frame link prediction in TKG as a sequence prediction problem. To handle this problem, we propose Recurrent Event Network (RE-NET) to model temporal, multi-relational, and concurrent interactions between entities. We study various kinds of aggregators, and evaluate our proposed work with different aggregators. Our extensive experiments show the effectiveness of RE-NET on predicting unseen relationships over time on two TKG datasets. Interesting future work includes modeling lasting events and performing joint inference for different subject-relation pairs.

References

[1] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2015.

[2] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. F. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, Çaglar Gülçehre, F. Song, A. J. Ballard, J. Gilmer, G. E. Dahl, A. Vaswani, K. R. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu. Relational inductive biases, deep learning, and graph networks. *CoRR*, abs/1806.01261, 2018.

[3] A. Bordes, N. Usunier, A. García-Durán, J. Weston, and O. Yakhnenko. Translating embeddings for modeling multi-relational data. In *NIPS*, 2013.

[4] E. Boschee, J. Lautenschlager, S. O'Brien, S. Shellman, J. Starz, and M. Ward. Icews coded event data. *Harvard Dataverse*, 12, 2015.

[5] K. Cho, B. van Merrienboer, Çaglar Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *EMNLP*, 2014.

[6] S. S. Dasgupta, S. N. Ray, and P. Talukdar. Hyte: Hyperplane-based temporally aware knowledge graph embedding. In *EMNLP*, 2018.

[7] T. Dettmers, P. Minervini, P. Stenetorp, and S. Riedel. Convolutional 2d knowledge graph embeddings. In *AAAI*, 2018.

[8] L. Du, Y. Wang, G. Song, Z. Lu, and J. Wang. Dynamic network embedding : An extended approach for skip-gram based network embedding. In *IJCAI*, 2018.

[9] A. García-Durán, S. Dumancic, and M. Niepert. Learning sequence encoders for temporal knowledge graph completion. In *EMNLP*, 2018.

[10] W. L. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *NIPS*, 2017.

[11] X. Han, S. Cao, X. Lv, Y. Lin, Z. Liu, M. Sun, and J.-Z. Li. Openke: An open toolkit for knowledge embedding. In *EMNLP*, 2018.

[12] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016.

[13] J. Leblay and M. W. Chekol. Deriving validity time in knowledge graph. In *Companion of the The Web Conference 2018 on The Web Conference 2018*, pages 1771–1776. International World Wide Web Conferences Steering Committee, 2018.

[14] K. Leetaru and P. A. Schrodt. Gdelt: Global data on events, location, and tone, 1979–2012. In *ISA annual convention*, volume 2, pages 1–49. Citeseer, 2013.

[15] F. Mahdisoltani, J. A. Biega, and F. M. Suchanek. Yago3: A knowledge base from multilingual wikipedias. In *CIDR*, 2014.

[16] G. H. Nguyen, J. B. Lee, R. A. Rossi, N. K. Ahmed, E. Koh, and S. Kim. Continuous-time dynamic network embeddings. In *WWW*, 2018.

[17] R. B. Palm, U. Paquet, and O. Winther. Recurrent relational networks. In *NeurIPS*, 2018.

[18] A. Pareja, G. Domeniconi, J. Chen, T. Ma, T. Suzumura, H. Kanezashi, T. Kaler, and C. E. Leisersen. Evolvegcn: Evolving graph convolutional networks for dynamic graphs. *CoRR*, abs/1902.10191, 2019.

[19] A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. A. Riedmiller, R. Hadsell, and P. W. Battaglia. Graph networks as learnable physics engines for inference and control. In *ICML*, 2018.

[20] M. S. Schlichtkrull, T. N. Kipf, P. Bloem, R. van den Berg, I. Titov, and M. Welling. Modeling relational data with graph convolutional networks. In *ESWC*, 2018.

[21] Y. Seo, M. Defferrard, P. Vandergheynst, and X. Bresson. Structured sequence modeling with graph convolutional recurrent networks. In *ICONIP*, 2017.

[22] R. Trivedi, H. Dai, Y. Wang, and L. Song. Know-evolve: Deep temporal reasoning for dynamic knowledge graphs. In *ICML*, 2017.

[23] R. Trivedi, M. Farajtabar, P. Biswal, and H. Zha. Dyrep: Learning representations over dynamic graphs. In *ICLR 2019*, 2019.

[24] T. Trouillon, J. Welbl, S. Riedel, É. Gaussier, and G. Bouchard. Complex embeddings for simple link prediction. In *ICML*, 2016.

[25] C. Xiong, S. Merity, and R. Socher. Dynamic memory networks for visual and textual question answering. In *ICML*, 2016.

[26] C. Xiong, V. Zhong, and R. Socher. Dynamic coattention networks for question answering. *CoRR*, abs/1611.01604, 2017.

[27] B. Yang, W. tau Yih, X. He, J. Gao, and L. Deng. Embedding entities and relations for learning and inference in knowledge bases. *CoRR*, abs/1412.6575, 2015.

[28] J. You, R. Ying, X. Ren, W. Hamilton, and J. Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *International Conference on Machine Learning*, pages 5694–5703, 2018.

[29] L. Zhu, D. Guo, J. Yin, G. V. Steeg, and A. Galstyan. Scalable temporal latent space inference for link prediction in dynamic social networks. *IEEE Transactions on Knowledge and Data Engineering*, 28:2765–2777, 2016.

A Details of Sequential Event Prediction in RE-NET

Transition from Equation (1) to (2) in the Main Paper. In Sec 3.1 of the main paper, we formalize the joint probability of TKG, also denoted by $\mathcal{E}$, as

$$\mathbb{P}(\mathcal{E}) = \prod_{t=1}^{T} \mathbb{P}(\mathcal{E}_t|\mathcal{E}_{t-1}, \ldots, \mathcal{E}_1)$$

Then we impose the assumption that $\mathcal{E}_t$ follows Markov assumption, *i.e.*, $\mathbb{P}(\mathcal{E}_t|\mathcal{E}_1, \ldots, \mathcal{E}_{t-1}) = \mathbb{P}(\mathcal{E}_t|\mathcal{E}_{t-1}, \ldots, \mathcal{E}_{t-m-1})$, and concurrent events occurring at time t are conditionally independent to each other given the past m event sets.

First, we formalize the joint probability of the TKG as $\mathbb{P}(\mathcal{E}) = \prod_{t=1}^{T} \mathbb{P}(\mathcal{E}_t|\mathcal{E}_{t-1}, \ldots, \mathcal{E}_1)$, and assume that $\mathcal{E}_t$ follows Markov assumption, *i.e.*, $\mathbb{P}(\mathcal{E}_t|\mathcal{E}_{t-1}, \ldots, \mathcal{E}_{t-m-1})$. Next, we assume that concurrent events occurring within the same time stamp t are *conditionally independent* to each other given the past m event sets. With the above assumptions, the joint probability of TKG at t can be rewritten into the following form:

$$\mathbb{P}(\mathcal{E}_t|\mathcal{E}_{t-1}, \ldots, \mathcal{E}_{t-m-1}) = \prod_{E_t \in \mathcal{E}_t} \mathbb{P}(E_t|\mathcal{E}_{t-1}, \ldots, \mathcal{E}_{t-m-1}).$$

With $E_t = (s, r, o, t)$ or equally (s_t, r_t, o_t), we can rewrite the above equation into the form as follows.

$$\mathbb{P}(E_t|\mathcal{E}_{t-1}, \ldots, \mathcal{E}_{t-m-1}) = \mathbb{P}(o_t|s_t, r_t, \{\mathcal{E}_{t-1}, \ldots, \mathcal{E}_{t-m-1}\})\mathbb{P}(s_t, r_t|\mathcal{E}_{t-1}, \ldots, \mathcal{E}_{t-m-1}).$$

In this work, we assume that probability of (s_t, r_t) is independent of the past m event sets $\{\mathcal{E}_{t-1}, \ldots, \mathcal{E}_{t-m-1}\}$, and model it using uniform distribution, leading to $\mathbb{P}(E_t|\mathcal{E}_{t-1}, \ldots, \mathcal{E}_{t-m-1}) \propto \mathbb{P}(o_t|s_t, r_t, \{\mathcal{E}_{t-1}, \ldots, \mathcal{E}_{t-m-1}\})$. Given (s, r), we assume that $\mathbb{P}(o_t|s_t, r_t, \{\mathcal{E}_{t-1}, \ldots, \mathcal{E}_{t-m-1}\})$ depends only on events that are related to s, and thus focus on the following conditional probability:

$$\mathbb{P}(o_t|s, r, \{\mathcal{E}_{t-1}, ..., \mathcal{E}_{t-m-1}\}) = \mathbb{P}(o_t|s, r, \{O_{t-1}(s), ..., O_{t-m-1}(s)\}),$$

where $\mathcal{E}_{t_i}$ becomes $O_{t_i}(s)$ which is a set of object entities interacted with subject entity s under *all* relations at time stamp t_i. We ignore the time index of (s, r) for simplicity in the rest of the paper.

B Event Sequence Encoder in Recurrent Event Network

We define an event sequence encoder based on RNN as follows:

$$h_t(s, r) = \text{RNN}(e_s, e_r, g(O_t(s)), h_{t-1}(s, r)).$$

We use Gated Recurrent Units [5] as RNN:

$$\begin{aligned}
a_t &= [e_s; e_r; g(O_t(s))] \\
z_t &= \sigma(W_z a_t + U_z h_{t-1}) \\
r_t &= \sigma(W_r a_t + U_r h_{t-1}) \\
h_t &= (1 - z_t) \circ h_{t-1} + z_t \circ \tanh(W_h a_t + U_h(r_t \circ h_{t-1})),
\end{aligned}$$

where ; is concatenation, $\sigma(\cdot)$ is an activation function, and $\circ$ is a Hadamard operator. The input is a concatenation of three vectors: subject embedding, object embedding, and aggregation of neighborhood representations.

C Dataset

We use four datasets: 1) two event-based temporal knowledge graphs (ICEWS18 and GDELT), and 2) two knowledge graphs (WIKI and YAGO). ICEWS is collected from 1/1/2018 to 10/31/2018, and GDELT is from 1/1/2018 to 1/31/2018.

WIKI and YAGO datasets have temporally associated facts $(s, r, o, [t_s, t_e])$. We preprocess the datasets such that each fact is converted to $\{(s, r, o, t_s), (s, r, o, t_s + 1_t), ..., (s, r, o, t_e)\}$ where 1_t is a unit time to ensure each fact has a sequence of events. Noisy events of early years are removed (before 1786 for WIKI and 1830 for YAGO).

The difference between the first group and the second group is that facts happen multiple times (even periodically) on the first group (event-based knowledge graphs) while facts last long time but are not likely to occur multiple times on the second group.

Dataset statistics are described on table 3.

Table 3: Dataset Statistics.

Data	N_{train}	N_{valid}	N_{test}	N_{ent}	N_{rel}	Time granularity
ICEWS18	373,018	45,995	69,514	23,033	256	24 hours
GDELT	1,734,399	238,765	305,241	7,691	240	15 mins
WIKI	539,286	67,538	63,110	12,554	24	1 year
YAGO	161,540	19,523	20,026	10,623	10	1 year

D Detailed Experimental Settings

Experimental Settings for Baseline Methods. In this section, we provide detailed settings for baselines. We use implementations of TransE and DistMult[5]. We implemented TTransE, TA-TransE, and TA-DistMult based on the implementation of TransE and Distmult, respectively. For TA-TransE and TA-DistMult, We use temporal tokens with the vocabulary of year, month and day on the ICEWS dataset and the vocabulary of year, month, day, hour and minute on the GDELT dataset. We use a margin-based ranking loss with L1 norm for TransE and TA-TransE and use a binary cross entropy loss for DistMult and TA-DistMult. We validate the embedding size among 100 and 200. We set batch size to 1024, margin to 1.0, negative sampling ratio to 1, and use the Adam optimizer.

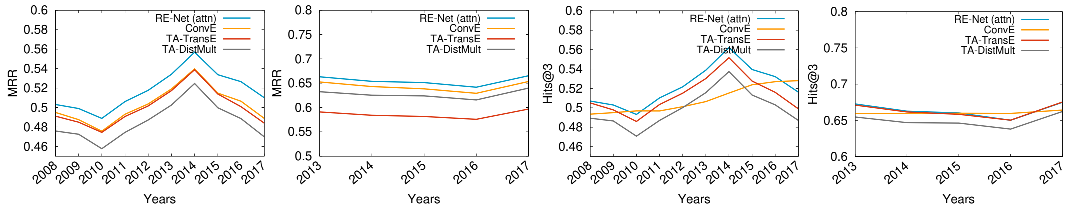
We use the implementation of ComplEx[6] [11]. We validate the embedding size among 50, 100 and 200. The batch size is 100, the margin is 1.0, and the negative sampling ratio is 1. We use the Adagrad optimizer.

We use the implementation of HyTE[7]. We use every timestamp as a hyperplane. The embedding size is set to 128, the negative sampling ratio to 5, and margin to 1.0. We use time agnostic negative sampling (TANS) for entity prediction, and the Adam optimizer.

We use the codes for ConvE[8] and use implementation by Deep Graph Library[9]. Embedding sizes are 200 for both methods. We use 1 to all negative sampling for ConvE and use 10 negative sampling ratio for RGCN, and use the Adam optimizer for both methods. We use the codes for Know-Evolve[10]. For Know-Evolve, we fix the issue in their codes. Issues are described in Section F. We follow their default settings.

Multi-step Inference over Time. To do multi-step predictions over time, we rank the candidate entities and select top-k entities as current predictions. We maintain the history as a sliding window of length p, so the oldest interaction set will be detached and new predicted entity set will be added to the history. So each entity has a list of history $\{O_{t-1}(s), ..., O_{t-m-1}(s)\}$. When RE-NET predicts entities at time t with relation r, we select top-k entities and store them into $O_t(s)$ with relation r. Here, we need to store the relation r together since this information is necessary to make a (predicted) knowledge graph at time t. When predictions of RE-NET are finished, we get the updated list of history $\{O_t(s), ..., O_{t-m-2}(s)\}$.

[5] https://github.com/jimmywangheng/knowledge_representation_pytorch
[6] https://github.com/thunlp/OpenKE
[7] https://github.com/malllabiisc/HyTE
[8] https://github.com/TimDettmers/ConvE
[9] https://github.com/dmlc/dgl/tree/master/examples/pytorch/rgcn
[10] https://github.com/rstriv/Know-Evolve



(a) WIKI (MRR) (b) YAGO (MRR) (c) WIKI (H@3) (d) YAGO (H@3)

Figure 5: Performance comparison of link prediction over different time stamps (i.e., temporal reasoning) on the test sets of WIKI and YAGO datasets with filtered metrics.

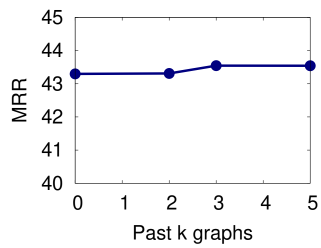


Figure 6: Performance on cumulative graphs.

E Additional Experiments

Performance of Prediction over Time. Figs. 5 shows the performance comparisons over different time stamps on the WIKI and YAGO datasets with filtered metrics. RE-NET (attn) outperforms other baselines with the MRR metric on both datasets. However, RE-NET (attn) and TA-TransE show similar performances with the Hits@3 metric. We notice that ConvE shows distinct performances with the Hits@3 metric. We suppose that ConvE is not affected by the time, which is why it shows different results.

Cumulative Graphs on RGCN aggregator. At each time stamp, multi-relational data builds a graph and a cumulative graph can be built through multi time stamps. We investigate performance using the cumulative graphs on RGCN aggregator of RE-NET. RGCN aggregator aggregates the representations of neighbors across current and past k time stamps. Fig. 6 shows the performances with current and past k cumulative graphs on the ICEWS18 dataset. If we use past k graphs, then the performances are slightly getting better. Cumulative graphs are denser and has more information about each node's local graph structure. Using cumulative graphs can be viewed as using a longer history. We see that the longer history over 5 in Fig. 3a of the main paper does not improve the performances. Long history already includes information from the past graph structure, which is why using additional past graphs does not improve the performances.

F Implementation Issues of Know-Evolve

We found a problematic formulation in the Know-Evolve model and codes. The intensity function (equation 3 in [22]) is defined as $\lambda_r^{s,r}(t|\bar{t}) = f(g_r^{s,r}(\bar{t}))(t - \bar{t})$, where $g(\cdot)$ is a score function, t is current time, and $\bar{t}$ is the most recent time point when either subject or object entity was involved in an event. This intensity function is used in inference to rank entity candidates. However, they don't consider concurrent event at the same time stamps, and thus $\bar{t}$ will become t after one event. For example, we have events $e_1 = (s, r, o_1, t_1), e_2 = (s, r, o_2, t_1)$. After e_1, $\bar{t}$ will become t (subject s's most recent time point), and thus the value of intensity function for e_2 will be 0. This is problematic in inference since if $t = \bar{t}$, then the intensity function will always be 0 regardless of entity candidates. In inference, all object candidates are ranked by the intensity function. But all intensity scores for all candidates will be 0 since $t = \bar{t}$, which means all candidates have the same 0 score. In their code, they give the highest ranks (first rank) for all entities including the ground truth object in this case.

Thus, we fixed their code for a fair comparison; we give an average rank to entities who have the same scores.

G Proofs and Analysis for Section

G.1 Proof for Theorem 1

Using pooling aggregator of GraphSAGE, we can actually copy its behavior by setting recurrent weight matrix of the RNN model to be 0. In this case, we lose all time-dependency our RE-NET and the representation model becomes *time-invariant*. However, RE-NET have exactly the same model capacity as GraphSAGE.

G.2 Analysis for Observation 1

Here we define the generation process of our temporal graph. Assume that the generation process of the graph follows a stochastic block model, and there are two communities in the graph. Half of the nodes belong to community A and the other half belong to community B. Nodes within one community have probability p_s to be connected while other pairs have $p_d < p_s$ probability to be connected. The edges in the graph are introduced into the graph in a time. Suppose a sequence of time-steps, a new node is introduced to the community and each edge is added to the graph.

This observation follows from three facts: (1) For each node v_j in the neighborhood $\mathcal{N}(v)$, using pooling aggregator, we can detect their community assignment s_j. We assign the output of community A to be $+1$ and the output of community B to be -1. (2) The error of incorrectly discerning the community of a node decrease exponentially with the number of links. For example let the node v be in community A. Let the total number of nodes at time t to be n_t, by Hoeffding's inequality we have

$$\mathbb{P}(\sum_{j:v_j \in \mathcal{N}(j)^t} s_j < 0) < exp(-2(p_s - p_d)^2 |\mathcal{N}(j)^t|)$$

(3) Given the correct community classification, the relation classifier is able to predict the probability of linking nodes.

Combining these three facts, RE-NET is able to infer the community structure of the node.