Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Event-centric Forecasting on Temporal Knowledge Graph with Transformer

Semester Thesis

Xiaoying Zhi
xiazhi@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Zhao Meng
Prof. Dr. Roger Wattenhofer

May 5, 2021

Acknowledgements

I want to thank my supervisor, Mr. Zhao Meng, for continuously supporting my project with his knowledge, experiences, available resources, and precious time. My thanks extend to Dr. Yunpu Ma and Mr. Zhen Han from Siemens AG Munich for their constant guidance in forming and keeping up with project plans and carrying out research ideas. I would also like to thank Prof. Roger Wattenhofer and DISCO group for providing sufficient academic and computing resources, which largely facilitated the project's progress.

Abstract

The project focused on event forecasting on the temporal knowledge graph (tKG), based on Hawkes process theories. We implemented a transformer-based model, inspired by the usage of RNN for the same task and the implementation of transformer for temporal events encoding under Hawkes process assumption. We achieved comparable performances with the transformer structure to that of RNN on the task. We also conducted further experiments and discussions on hyperparameter tuning and modifications on the graph information integration method and hidden state decoding based on Hawkes process.

Contents

Chapter 1

Introduction

The ability to predict the future is chased by humans, covering but not limited to areas such as the stock market, economic development, weather forecasting, and sometimes a person's fate. To better predict future events based on historical events, essential tasks include describing and encoding past events, recognizing regular patterns, and linking from past to future. Multiple scientific, mathematical, statistical, and computational tools are invented for these tasks.

A stochastic process model, Hawkes process [1], can be applied to describe the process of a sequence of events with temporal information. Hawkes process was proposed to describe a category of "self-exciting" events, where past events would affect the probability of future events' occurrence through an intensity function. An extension of Hawkes process to cover both "self-exciting" and "self-inhibitory" events is Neural Hawkes process [2], a multivariate point process implemented with a continuous LSTM. Later, Transformer Hawkes process [3] was proposed, which replaces the RNN with Transformer to better capture long-term dependencies. It has achieved better performances in learning events' temporal dependencies than Neural Hawkes Process.

Apart from serial temporal events, the temporal knowledge graph (tKG) is also a structure to store related events with temporal information. A tKG differs from a series of temporal events in the sense that events are stored under a graph structure, and each event is described by a quadruple: {subject, predicate/relation, object, time}. Given one event, the corresponding history sequential events could be extracted as the ones related to and have an occurrence time earlier than the query quadruple, e.g., all history events with the same object/subject and the same predicate. Common forecasting tasks on tKG include time prediction and event prediction, and the latter can include subject/object prediction and relation prediction. An efficient method to encode and decode events with implicit relational information is required to perform such predictions. Some outstanding works include TransE [4], ComplexE [5], RotatE [6], etc.

To enhance the performances on prediction tasks on tKG, Graph Hawkes Neural Network [7] was proposed, which manipulates the tKG structure to sequential temporal events, maps to a latent space by LSTM, and performs forecasting with the help of intensity function from Hawkes process. It has achieved state-of-the-art event prediction performances on two global event datasets, Global Database of Events, Language, and Tone (GDELT) [8], and Integrated Crisis Early Warning System (ICEWS) [9].

Based on the knowledge that transformer structure generally performs better on long-term in-sequence dependency than RNN models [10] and that with a delicate batch size setting [11], the transformer can achieve higher training efficiency than RNNs, we implemented transformer to temporal knowledge graph data for event-centric forecasting and conducted several related experiments. Our experiments are conducted on ICEWS data after preprocessing according to task requirements and model choice.

We prove that while having evaluating metrics approaching the RNN's, transformer's performances depend delicately on the settings of hyperparameters during training, and most of the time, they cannot surpass RNN. We will present the experiment results thoroughly, analyze, and give possible explanations for the transformer's inability to beat the RNN model.

Chapter 2

Background

2.1 Hawkes Process and Extensions

We will discuss Hawkes Process's background, why it is instructive in sequential events encoding and decoding, and how neural networks (more complex models) can be constructive in this task.

2.1.1 Hawkes Process

Mathematically, one can consider events in a sequence (stream) as independent or dependent in some ways. The mathematical field aiming at describing such event streams is called the point process. One most basic model for event sequences is Poisson process [12], which assumes that events are mutually independent, i.e., one event's occurrence would not affect other events' occurrences at all.

Hawkes process is another mathematical model in point process that assumes past events will affect the probability of future events' happening and models past events' influences to future events. It is widely applied to geological events, financial contagions, high-frequency trading activities [13], and similar areas.

One core concept of Hawkes process is the intensity function λ, representing the event arrival rate (or equivalently, how the previous events affect later events' arrival) through time. Traditionally, stationary point processes describe the intensity function as

$$\lambda = E[dN(t)]/dt, \tag{2.1}$$

where $N(t)$ represents the number of events accumulated up to t, and which makes λ a constant, meaning an average rate of event arrival throughout time [1, 14].

On the other hand, Hawkes process introduces a type of "self-exciting" events, where the arrival of later events are determined by the summed influences of all previous events. In terms of intensity function, it is described as

$$\lambda = E[\Lambda(t)] = \nu + \lambda \int_{-\infty}^{t} g(t-u)du, \tag{2.2}$$

and equivalently,

$$\lambda = \nu / \{1 - \int_{0}^{\infty} g(v)dv\}, \tag{2.3}$$

where $\Lambda(t)$ is a stationary random process, ν can be seen as the current event's intensity by itself (or base rate, more formally), and the integral of g's are the summed influences of past events. With stationarity assumption of events in Hawkes process, there must be $\nu > 0$ and $\int_0^\infty g(v)dv < 1$.

Additionally, in Hawkes process's original statements, $g(v)$ is a non-negative function such that,

$$g(v) \begin{cases} \geq 0 & \text{if } v \geq 0 \\ = 0 & \text{if } v < 0. \end{cases} \tag{2.4}$$

One special case that satisfies this non-negativity is the exponential decay, where

$$g(v) := \sum_{j=1}^{k} \alpha_j e^{-\beta_j v} (v > 0) \tag{2.5}$$

which is a combination of multiple functions in the exponential family. The exponential decay is adopted widely in applications of Hawkes process.

However, the non-negativity restriction on $g(v)$ is removed in Neural Hawkes Process, to describe "self-inhibitory" events. Accordingly, the restriction of $v > 0$ in Eq.2.5 is removed as well as that of $\int_0^\infty g(v)dv < 1$ after Eq.2.3.

2.1.2 Neural Hawkes Process

Firstly proposed in 2017, Neural Hawkes Process [2] intended to extend the conventional Hawkes Process by constructing a "neurally self-modulating multivariate point process", namely introducing an LSTM to the process of computing the intensity λ of events. The former restriction on intensity's non-negativity is omitted with such a setting, which allows for the description of "self-inhibitory" events with the model. A graphical illustration of self-exciting and -inhibitory events is shown in Fig. 2.1.

The non-negativity restriction is abandoned based on the compelling influences between real events, to which we apply the model. One compelling example is that on individuals, cake consumption can potentially inhibit cookie assumption. Additionally, the exponential decay assumption can also be violated to cover more complex situations of delayed decay or a switch in excitation/inhibition during two events' time intervals.

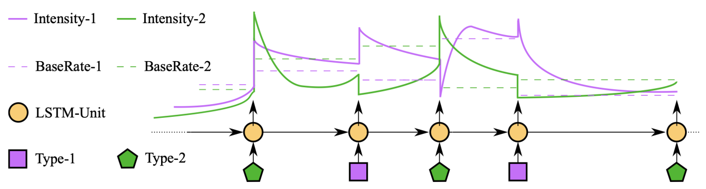


Figure 2.1: An illustration on self-exciting and self-inhibitory events [2].

Since the exponential decay assumption is omitted, another rule to fill in the continuous-time gap between two discrete events must be introduced. For Neural Hawkes Process, this is realized by a continuous-LSTM, which has the same structure as conventional LSTMs. The exponential decay in Hawkes process still applies to the gap between, but with respect to a non-constant function, modeling the aforementioned delayed and switched exciting/inhibitory effects becomes possible.

Neural Hawkes Process denotes an event as a tuple with two elements (k, t), where $k \in \{1, 2, ..., K\}$ is the event type, and $t \in \mathbb{R}$ is the occurrence time of the event. An event stream is then a sequence of such tuples $(k_1, t_1), (k_2, t_2), ...$ The sequence can be theoretically infinitely long, but in reality, often with a finite length.

Eq.2.6 shows the updating rules of continuous-LSTM.

$$\mathbf{i}_{m+1} \leftarrow \sigma(\mathbf{W}_i \mathbf{k}_m + \mathbf{U}_i \mathbf{h}(t_m) + \mathbf{d}_i) \tag{2.6a}$$

$$\bar{\mathbf{i}}_{m+1} \leftarrow \sigma(\mathbf{W}_{\bar{i}} \mathbf{k}_m + \mathbf{U}_{\bar{i}} \mathbf{h}(t_m) + \mathbf{d}_{\bar{i}}) \tag{2.6b}$$

$$\mathbf{f}_{m+1} \leftarrow \sigma(\mathbf{W}_f \mathbf{k}_m + \mathbf{U}_f \mathbf{h}(t_m) + \mathbf{d}_\mathbf{f}) \tag{2.6c}$$

$$\bar{\mathbf{f}}_{m+1} \leftarrow \sigma(\mathbf{W}_{\bar{f}} \mathbf{k}_m + \mathbf{U}_{\bar{f}} \mathbf{h}(t_m) + \mathbf{d}_{\bar{\mathbf{f}}}) \tag{2.6d}$$

$$\mathbf{z}_{m+1} \leftarrow 2\sigma\left(\mathbf{W}_z \mathbf{k}_m + \mathbf{U}_z \mathbf{h}\left(t_m\right) + \mathbf{d}_z\right) - 1 \tag{2.6e}$$

$$\mathbf{o}_{m+1} \leftarrow \sigma\left(\mathbf{W}_\mathrm{o} \mathbf{k}_m + \mathbf{U}_\mathrm{o} \mathbf{h}\left(t_m\right) + \mathbf{d}_\mathrm{o}\right) \tag{2.6f}$$

$$\mathbf{c}_{m+1} \leftarrow \mathbf{f}_{m+1} \star \mathbf{c}\left(t_m\right) + \mathbf{i}_{m+1} \star \mathbf{z}_{m+1} \tag{2.6g}$$

$$\bar{\mathbf{c}}_{m+1} \leftarrow \bar{\mathbf{f}}_{m+1} \odot \bar{\mathbf{c}}_m + \bar{\imath}_{m+1} \odot \mathbf{z}_{m+1} \tag{2.6h}$$

$$\boldsymbol{\delta}_{m+1} \leftarrow f\left(\mathbf{W}_\mathrm{d} \mathbf{k}_m + \mathbf{U}_\mathrm{d} \mathbf{h}\left(t_m\right) + \mathbf{d}_\mathrm{d}\right) \tag{2.6i}$$

The $f(\cdot)$ that calculates the decay function $\boldsymbol{\delta}$ is a softplus function, a smoothed ReLU:

$$f(x) = \psi \log(1 + \exp(x/\psi)), \tag{2.7}$$

and $\mathbf{h}(t)$ is computed from the LSTM variables as in Eq.2.9. The intensity function is K-dimensional, referring to the K types of events, in favor of event type prediction.

$$\lambda_k(t) = f_k\left(\mathbf{w}_k^\top \mathbf{h}(t)\right) \tag{2.8}$$

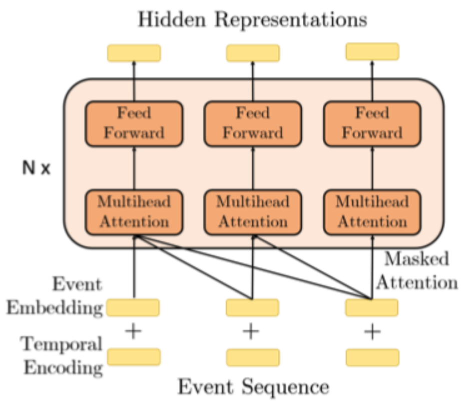


Figure 2.2: An illustration on the architecture of Transformer Hawkes Process [3]

$$\mathbf{h}(t) = \mathbf{o}_i \odot (2\sigma(2\mathbf{c}(t)) - 1) \text{ for } t \in (t_{i-1}, t_i] \tag{2.9}$$

The continuous characteristic in the continuous-LSTM can be observed in the difference between discrete $\mathbf{c}_i$, $\bar{\mathbf{c}}_\mathbf{i}$ in Eq.2.6 and continuous $\mathbf{c}_t$ in Eq.2.9. The transition from discretion to continuity is done similarly to exponential decay in Hawkes process:

$$\mathbf{c}(t) \overset{\text{def}}{=} \bar{\mathbf{c}}_{m+1} + (\mathbf{c}_{m+1} - \bar{\mathbf{c}}_{m+1}) \exp\left(-\boldsymbol{\delta}_{m+1}(t - t_m)\right) \text{ for } t \in (t_m, t_{m+1}]. \tag{2.10}$$

We will show later that the $\mathbf{h}_t$ can be extended to other types of neural networks, and the inputs of the neural network can also be extended to contain richer information.

2.1.3 Transformer Hawkes Process

The key idea of Transformer Hawkes Process [3] is to replace the RNN in Neural Hawkes Process with a transformer structure for theoretically better performances in long-term dependency extraction as well as higher computational efficiency. Fig.2.2 is a graph illustration of the model architecture.

Two significant differences between the transformer and neural Hawkes process are that: 1) The intensity function in the transformer structure is expanded to contain information beyond hidden states, but also time; 2) The hidden states are calculated from the transformer's encoder, which is the result of matrix computations taking the sequence as a whole, instead of updating on every event as in RNN.

The intensity function is calculated from:

$$\lambda_k(t) = f_k\left(\alpha_k \frac{t - t_i}{t_i} + \mathbf{w}_k^\top \mathbf{h}(t_i) + b_k\right) \text{ for } t \in (t_{i-1}, t_i], \tag{2.11}$$

from which we can observe enriched information directly from current event time and base rate. The f_k is also chosen to be a scaled softplus function.

The transformer's encoder calculates the hidden state $\mathbf{h}(t)$ as in Eq.2.13. $\mathbf{U} \in \mathbb{R}^{M\times K}$ is the embedding matrix for all K event types in the transformer settings, $\mathbf{Y} = [\mathbf{k}_1, \mathbf{k}_2, \ldots, \mathbf{k}_L] \in \mathbb{R}^{K\times L}$ is the collection of event type embedding, and $\mathbf{Z} = [\mathbf{z}(t_1), \mathbf{z}(t_2), \ldots, \mathbf{z}(t_L)] \in \mathbb{R}^{M\times L}$ is the collection of event time encodings. The event time encoding is calculated by

$$[\mathbf{z}(t_j)]_i = \begin{cases} \cos\left(t_j / 10000^{\frac{i-1}{M}}\right), & \text{if } i \text{ is odd} \\ \sin\left(t_j / 10000^{\frac{i}{M}}\right), & \text{if } i \text{ is even} \end{cases} \tag{2.12}$$

$$\mathbf{X} = (\mathbf{U}\mathbf{Y} + \mathbf{Z})^\top \tag{2.13a}$$

$$\mathbf{Q}_h = \mathbf{X}\mathbf{W}_h^Q, \quad \mathbf{K}_h = \mathbf{X}\mathbf{W}_h^K, \quad \mathbf{V}_h = \mathbf{X}\mathbf{W}_h^V \tag{2.13b}$$

$$\mathbf{S}_h = \text{Softplus}\left(\frac{\mathbf{Q}_h \mathbf{K}_h^\top}{\sqrt{M_K}}\right)\mathbf{V}_h \tag{2.13c}$$

$$\mathbf{S} = [\mathbf{S}_1, \mathbf{S}_2, \ldots, \mathbf{S}_H]\,\mathbf{W}^O \tag{2.13d}$$

$$\mathbf{H} = \text{ReLU}\left(\mathbf{S}\mathbf{W}_1^{\text{FC}} + \mathbf{b}_1\right)\mathbf{W}_2^{\text{FC}} + \mathbf{b}_2 \tag{2.13e}$$

$$\mathbf{h}(t_i) = \mathbf{H}(i, :) \tag{2.13f}$$

Chapter 3 will explain a similar structure that can apply to event-centric temporal knowledge graph forecasting.

2.2 Temporal Knowledge Graph

This section will discuss the temporal knowledge graph (tKG), its structure characteristics, and relevant past works on embedding and forecasting.

2.2.1 Semantic Knowledge Graph

The semantic knowledge graph is a kind of multi-relational knowledge base that stores factual information. One graph illustration is as Fig.2.3. Each node in the graph represents an entity, and each edge in the graph represents a relation/predicate. The knowledge graph defines an event with two nodes and one connecting edge, namely two entities (subject and object) and one relation. It denotes an event in the form of a tuple of three elements: (s, r, o), pointing at subject, predicate, object, respectively.

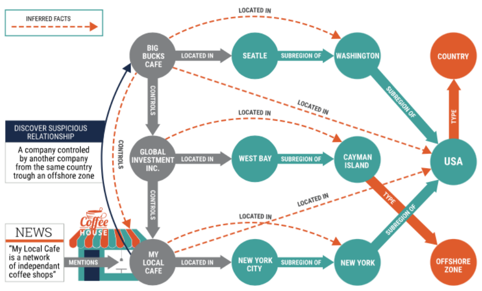


Figure 2.3: A graphic illustration of semantic knowledge graph [15]

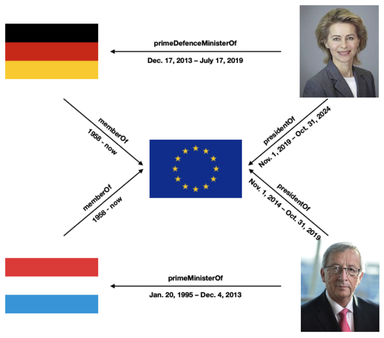

primeDefenceMinisterOf
Dec. 17, 2013 – July 17, 2019
memberOf
1958 - now
presidentOf
Nov. 1, 2019 – Oct. 31, 2024
memberOf
1958 - now
presidentOf
Nov. 1, 2014 – Oct. 31, 2019
primeMinisterOf
Jan. 20, 1995 – Dec. 4, 2013

Figure 2.4: A graphic illustration of temporal knowledge graph [7]

2.2.2 Temporal Knowledge Graph

A temporal knowledge graph (tKG) is a semantic knowledge graph with enriched information in time; namely, each edge simultaneously contains the relation and the time, the latter indicating the valid period/timestamp of the relation. An example is Fig.2.4. Accordingly, each event in tKG shall contain two nodes, one connecting edge, and multiple discrete time stamps (discretion only in favor of sampling and computation). Formally, a quadruple of four elements: (s, r, o, t) should be involved, pointing at the subject, relation, object, and time, respectively.

One can consider a sequence of such events as an event stream previously discussed in Section 2.1.2. There exists an information gap between (k, t) and (s, r, o, t). One can technically take the combination (s, r, o) as individually categorized events, but that would impose $\mathcal{O}(K_s \cdot K_r \cdot K_o)$ types of events in total, potentially heavy for computation. Conventionally, in forecasting problems, researchers usually focus on only one element, e.g. predicting the type of s given (r, o, t). Chapter 3 will include more discussions on how to incorporate the information from the triplet.

Although predicting t (time prediction) is also an interesting task in the academia, this thesis will only focus on event-centric predictions and not on time predictions.

2.2.3 Temporal Knowledge Graph Completion

One common task on knowledge graph is graph completion, equivalent to link (entity/relation) prediction. On static (semantic) KGs, common approaches include introducing a scoring function w.r.t. triplets (s, r, o) and perform classification on candidate classes based on the scores [16]. The scoring functions are computed from learnable embeddings of links. Scoring Examples include TransE [4] (Eq.2.14), DistMult [17] (Eq.2.15), etc.

$$f(s, r, o) = ||\mathbf{e}_s + \mathbf{e}_r - \mathbf{e}_o||_2 \tag{2.14}$$

$$f(s, r, o) = (\mathbf{e}_o \star \mathbf{e}_s)\mathbf{e}_r^{\intercal} \tag{2.15}$$

Various encoding methods for the time information in tKGs can apply to neural network models. Garcıa-Duran et al. [16] split time into tokens, then input the tokens to an LSTM to generate a latent representation. For example, the year stamp "2009" is split to "2","0","0",and "9", and formed as part of the input sequence of the LSTM. On the other hand, Han et al. [7] directly used event time to calculate the time-dependent exponential decay in the intensity function of the point process. The intensity function of each candidate link class will be transformed to a probability, based on which the neural network makes a prediction. More details will be in the following section.

Another task on temporal knowledge graph is time prediction, i.e., given an event, predicting its occurrence time. It is still a less discussed topic, so is it in this thesis.

2.3 Graph Hawkes Neural Network

Graph Hawkes Neural Network [7], short as GHNN, is an RNN- and Hawkes process-based model for temporal knowledge graph forecasting, including link prediction and time prediction. It consists of three major parts: mean aggregator, continuous LSTM, and predictor. A model architecture illustration is in Fig.2.5.

Event quadruples (s, r, o, t) are firstly grouped to form event streams if they share the same (s, r) or (r, o), depending on whether the sub-task is object prediction or subject prediction respectively. Take the (s, r) scenario as an example. At each timestamp, several events share the same (s, r) tuple. The mean aggregator then calculates the mean of the embeddings $\mathbf{e}_o$ of all objects $\mathbf{O}$ in these

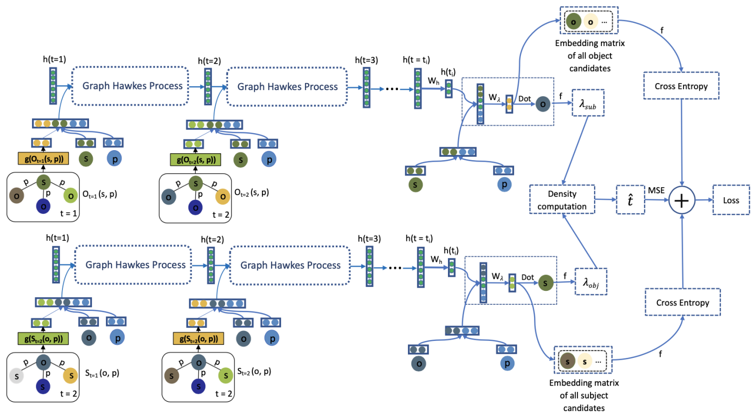


Figure 2.5: A graphic illustration of GHNN [7]. The upper and lower path can be regarded as object and subject prediction, respectively. The combination of two paths is part of the time prediction pipeline.

events. The aggregated tensor $g(\mathbf{O})$ is concatenated to the $\mathbf{e}_s$ and $\mathbf{e}_r$ as the input to the RNN:

$$\mathbf{k}_m\left(e_{s_i}, e_{r_i}, e_i^{h,sr}\right) = g\left(\mathbf{O}_{t_m}\left(e_{s_i}, e_{r_i}\right)\right) \oplus \mathbf{e}_{s_i} \oplus \mathbf{e}_{r_i}. \tag{2.16}$$

The RNN structure is continuous-LSTM (cLSTM) mentioned in Section 2.1.2. The update rules are as Eq.2.6, 2.10, and 2.9.

With the hidden states $\boldsymbol{h}(t)$ obtained, the intensity function of all candidate classes is calculated by:

$$\lambda\left(e_o \mid e_{s_i}, e_{p_i}, t_i, e_i^{h,sr}\right) = f\left(\mathbf{W}_\lambda\left(\mathbf{e}_{s_i} \oplus \mathbf{h}\left(e_o, e_{s_i}, e_{r_i}, t_i, e_i^{h,sr}\right) \oplus \mathbf{e}_{r_i}\right) \cdot \mathbf{e}_o\right), \tag{2.17}$$

where $\oplus$ represents concatenation, $\mathbf{e}_o$ is the entity embedding of all types, e_{s_i}, e_{r_i} are repsectively the subject and relation at time t_i, and f is the softplus function.

GHNN is proved to reach state-of-the-art performances on link prediction on temporal knowledge graphs.

Chapter 3

Methodology

We will discuss our task settings, model architecture and rationales, and more details on our model's components.

3.1 Task Description

The primary task involved is event-centric forecasting on temporal knowledge graph, in plain words, event (subject/object) type prediction given a sequence of historical events in a temporal knowledge graph. More formally, denoting the event sequence as a series of quadruples $(s_1, r_1, o_1, t_1), ..., (s_n, r_n, o_n, t_n)$, we try to predict s_{n+1} given $(r_{n+1}, o_{n+1}, t_{n+1})$, or similarly to predict o_{n+1} given $(s_{n+1}, r_{n+1}, t_{n+1})$.

In a large-scale knowledge graph, at each t, there can be a considerable amount of quadruples involved, and while some of them can be quite relevant to the queried event, many of them can also be irrelevant. There are commonly two methods to deal with this issue.

Firstly, we build the sequence so that if we query the subject in the form of $(?, r_q, o_q, t_q)$, we select the past events with the same relation and object to the event stream, i.e., $(s_i, r_q, o_q, t_{\leq t_q})$. Vice versa for object query. The number of involved events is thus limited, and more irrelevant information is omitted. In reality, the sequence is restricted to a finite maximum length.

Secondly, to integrate information from multiple events simultaneously, we introduce an aggregator, which compresses the information to a fixed-dimension tensor. More details are in Section 3.2.3.

3.2 Model Architecture

Our model mainly consists of an embedding block, an aggregator, and a transformer. The embedding block generates and learns embeddings of all entity and

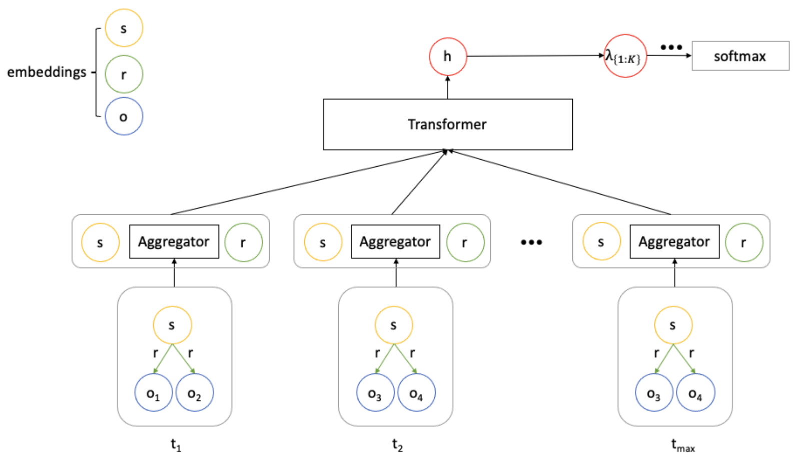


Figure 3.1: A graph illustration of our model architecture

relation types. The aggregator obtains a compressed and rich-in-information tensor from the preprocessed input event sequence. The transformer learns the sequence pattern and conducts predictions. Specifically, we only apply the transformer's encoder to obtain the hidden state without passing it to the decoder. A graph illustration is as in Fig.3.1.

Following are discussions on each block with more details.

3.2.1 Embeddings

In this model, both the parameters and the embeddings are learnable. The embeddings have a fixed size and randomly initialized values.

The subjects and objects share the same set of entity embeddings, covering K_e classes of entity types. The relation embedding for each event, more delicately, is different for the two directions: $s \to o$ and $o \to s$. Therefore, the relation embeddings cover $2K_r$ classes of relation types. In the implementation, the corresponding embedding group is selected according to the task's direction (to predict s given o or reversed).

3.2.2 Transformer

The transformer is the same as a conventional one without a decoder. We used the same structure as in Section 2.1.3 and Eq. 2.12 - 2.13.

The transformer's input is the term $\mathbf{X} \in \mathbb{R}^{L \times M}$ in Eq. 2.13, where each row represents an event, and each column corresponds to a dimension of the input embedding. As shown in Fig. 3.1, an event's input is a sum of 1) temporal encoding and 2) concatenation of subject embedding, aggregator output, and relation embedding. The temporal encoding is done through Eq. 2.12, imposing a mixed trigonometric function to the vector representing event times.

Two core modules of the transformer are the multi-head self-attention and feedforward network. Self-attention ensures that while the network takes the whole sequence as the input, it only sees the corresponding history sub-sequence at each output sequence position. Multi-head attention provides multiple sets of weights to the input, similar to the idea of model bootstrapping.

Referring back to Eq. 2.13, the attention module output $\mathbf{S}_h$ results from a computation rule applied to key, query and value, corresponding to $\mathbf{K}_h$, $\mathbf{Q}_h$, and $\mathbf{V}_h$, variables. Self-attention in implementation takes the input sequence as both key and query so that the attention is trained on the sequence itself. Specifically, we call the parameter $\sqrt{M_K}$ temperature of the softmax function or the transformer, which controls how close the softmax is to a max function. The lower the temperature, the closer it is to the max. Multi-head attention is done through a weighted aggregation of the outputs from multiple self-attention modules, which gives the variable $\mathbf{S}$.

Following the attention module is a pointwise feedforward network, denoted by $\mathbf{H}$. The hidden state of each event $\mathbf{h}_i$ is thus one column of the tensor $\mathbf{H}$.

After all, note that the sequence length can be different for each stream. Thus, a padding mask appears to ensure each tensor variable keeps a constant size throughout the transformer processing.

In our scenario, only the last event of each event stream is under interest. We, therefore, only take the last event's hidden state into the intensity calculation and prediction stage.

3.2.3 Aggregator

TKGs require an aggregator to compress the information from a group of entities to one representational vector. Two important types experimented here are the mean aggregator [7] and relational aggregator [18]. Other aggregators in past literature include the multi-hop aggregator [19], weighted aggregator [20], and many more.

The mean aggregator's output is calculated as:

$$g\left(\mathbf{O}_t\left(e_{s_i}, e_{r_i}\right)\right) = \frac{1}{\left|\mathbf{O}_t\left(e_{s_i}, e_{r_i}\right)\right|} \sum_{\mathbf{e}_o \in \mathbf{O}_t\left(e_{s_i}, e_{r_i}\right)} \mathbf{e}_o, \tag{3.1}$$

where $\mathbf{O}_t(e_{s_i}, e_{r_i})$ is the group of objects at time t from the events with the same s and r type, and $\mathbf{e}$ is the embedding of the entity/relation type. The aggregator can also apply to grouped subjects by simply switching o and s in the equation.

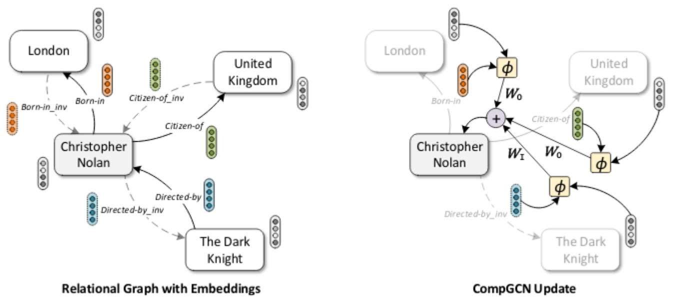


Figure 3.2: A graphical illustration of relational aggregator: a weighted sum of composition functions of entity and relation embeddings. [18]

The relational aggregator can be regarded as a weighted sum of entity and relation embeddings' composition operation, as illustrated in Fig.3.2. A composition operation $\phi(\cdot)$ can be either subtraction, multiplication, or circular-correlation:

$$\text{Subtraction: } \phi\left(e_s, e_r\right) = e_s - e_r \tag{3.2a}$$

$$\text{Multiplication: } \phi\left(e_s, e_r\right) = e_s * e_r \tag{3.2b}$$

$$\text{Circular-correlation: } \phi\left(e_s, e_r\right) = e_s \star e_r. \tag{3.2c}$$

In our settings, we experimented on the circular correlation only, in favor of its surpassing performance than the other two in the original work.

3.2.4 Intensity Function

Intensity function is an essential part of Hawkes process-related analysis. In GHNN [7], Eq.2.17 calculates the intensity function as a non-linearly transformed weighted function of a concatenation that includes the hidden state.

In Transformer Hawkes process [3], Eq.2.11 calculates the intensity function as a non-linearly transformed summation of time-related constant $\alpha_k \frac{t-t_i}{t_i}$, weighted hidden states $\mathbf{w}_k^\intercal \mathbf{h}(t_i)$, and a base rate constant b_k. The base rate constant is realized through a linear layer simultaneously with the weighting of hidden states.

Our model proposes a combination of the two – we replaced the hidden state in Eq.2.11 with the concatenation in Eq.2.17:

$$\lambda(e_{o_q} \mid h_i^{s,r}) = f\left((\alpha_t \frac{t_q - t_i}{t_i} + \mathbf{W}_\lambda(\mathbf{e}_s \oplus \mathbf{h}(e_{o_q}, h_i^{s,r}) \oplus \mathbf{e}_r) + \mathbf{b}_\lambda) \cdot \mathbf{e}_o\right), \tag{3.3}$$

where $h_i^{s,r}$ represents the history event sequence of event i, $\mathbf{h}(e_{o_q}, h_i^{s,r})$ is the output hidden state of the history from the transformer, and t_i is the time of the event, the same as the last event's time in the sequence. The function f is the softplus function.

The predicted entity class is the one with the highest λ score.

3.3 Learning Scheme

The learning objective is the cross-entropy loss:

$$\mathcal{L} = (\mathcal{L}^{\mathrm{s,r}} + \mathcal{L}^{\mathrm{o,r}})/2, \tag{3.4}$$

where $\mathcal{L}^{s,r}$ is the cross-entropy loss on object prediction given (s, r, t) tuples, and $\mathcal{L}^{o,r}$ is that on subject prediction, written as:

$$\mathcal{L}^{\mathrm{s,r}} = -\sum_{i=1}^{N}\sum_{k=1}^{K_e} y_k \log\left(p\left(e_{o_i} = k \mid h_i^{s,r}\right)\right) \tag{3.5a}$$

$$\mathcal{L}^{\mathrm{o,r}} = -\sum_{i=1}^{N}\sum_{k=1}^{K_e} y_k \log\left(p\left(e_{s_i} = k \mid h_i^{o,r}\right)\right). \tag{3.5b}$$

In the implementation, y_k is the one-hot label of event i at class k. The probability term $p(e_{o_i} = k \mid h_i^{s,r})$ is equivalent to the intensity function λ after normalization on the corresponding category.

Chapter 4

Experiments

Several experiments are conducted on our proposed model and show that our model reaches comparative performances as RNN. We have also performed additional experiments on several individual compositions and parameters. We will show their influences on the model's ability to predict.

4.1 Datasets and Preprocessing

4.1.1 Datasets

One commonly used family of temporal knowledge graph is ICEWS - Integrated Crisis Early Warning System [9], which contains events with the essence of coded interactions between socio-political actors. Such actors have nation-states, sectors, groups, or individuals, and such interactions include hostile or cooperative actions.

Two sub-datasets in the family, ICEWS14 and ICEWS0515, are used in the experiments. ICEWS14 contains all events that happened in 2014, and ICEWS0515 includes all events that happened in a 10-year span from 2005 to 2015. Hereunder are statistics of the two datasets:

Dataset	# Entity types	# Relation types	# Time stamps	Events
ICEWS14	12498	254	270	492679
ICEWS0515	10488	251	4017	461329

Table 4.1: Statistics of datasets

The ICEWS0515 dataset has a train, a validation, and a test set. The original ICEWS14 dataset, however, contains only train and test sets, without a validation set. Therefore, we made a resplitted version ICEWS14_resplit and adjusted the ratio of train:validation:test to 7:1.5:1.5, the same as ICEWS0515.

4.1.2 Preprocessing

The ICEWS dataset contains raw quadruples only. We extract two fixed-length history event streams for each event quadruple during preprocessing – one is composed of history quadruples with the same subject and relation and the other with the same relation and object. Appendix A shows the statistics of the generated sequences.

It is observed that in both ICEWS14 and ICEWS0515 datasets, the validation and test sequences have a slightly heavier distribution at longer lengths, which we assume the intention is to test the model's generalization ability to predict excessively far events. Such an uneven distribution can be meaningful on longer sequence lengths, but its utility is limited to a shorter maximum sequence length.

It is also observed that on all the datasets, the mean (and maximum) of the generated sequences' lengths are slightly higher in object-centric ones than subject-centric ones. For simplicity, we still take the average the two losses on subject- and object-centric sub-datasets. Nevertheless, it is also plausible to assign a higher weight on subject-centric loss to compensate for the averagely longer lengths in object-centric sequences, or vice versa.

We selected some fixed maximum history lengths when generating the sequences and have conducted experiments on each of them. The maximum acceptable length of the transformer follows the maximum history length of the preprocessed dataset.

4.2 Evaluating Metrics

We use two major metrics: mean reciprocal rank (MRR) and Hits@K, for their common appearance in past literature on (temporal) knowledge graphs [7, 4, 16, 21].

A reciprocal rank is the reciprocal of the true entity class's rank of probability among all classes. MRR is then the mean of all events' reciprocal ranks in a dataset.

Hits@K measures the percentage of events in a dataset whose true class falls under the top-K ranked categories in prediction. We take $K = 1, 3, 10$, aligned with [7].

For each predicted entity, we compute two types of ranks – raw and filtered, following GHNN [7]. The raw rank is the rank of the ground truth's intensity among that of all entity types. The filtered rank is the rank of the ground truth's intensity among all entity types, except the ones that are also the ground truth. Multiple ground truths can appear simultaneously since one pair of a subject and relation can have multiple objects simultaneously, and vice versa for one object and one relation. One example is that one country can be a business partner to many countries at the same time. The two types of ranks would give two results to MRR and Hits@K. Therefore, we present two sets of metrics, raw and filtered, on the test set.

4.3 Experimental Setup

Among all experiments, some fixed parameters are as follows:

Embedding dimensions: Entities and relations: 200; Aggregator output: 200; Temporal encoding: 600;

Optimizer: Adam [22]; Learning rate: 0.0001; Weight decay: 0.00001;

Dropout of layers not within transformer: 0.5

Softplus Scale β: 10.0

Transformer: Head number: 4; Encoder layer number: 4; Feedforward inner layer dimension: 1024; Output hidden layer size: 200.

We applied gradient accumulation [23] during training for some large batch sizes, i.e., to update the optimizer every few batches. We used automatic mixed precision during training to speed up the process and maximize storage usage for some large sequence lengths.

On the two datasets with a validation set, the model under test is the one with the highest MRR on the validation set. On ICEWS14 without a validation set, the model under test can slightly overfit to the training set. In the following sections, experiment results from ICEWS14 will be omitted if not necessary. Whenever necessary, the highest scores observed would be presented, and those that show a sign of overfitting would be discarded if possible.

4.4 Experiment Results

We will first present comparisons of our transformer model with GHNN [7], then discuss the function of each part and hyper-parameters in our model with parallel experiments.

4.4.1 Comparison with RNN

We modified GHNN's model to omit time prediction-related computations and loss to align with the task setting. Then GHNN and transformer are compared under the same common hyperparameters, e.g., batch size, learning rate, etc. We present the performances on different max sequence lengths.

Fig.4.1, 4.2, and 4.3 show the transformer model and GHNN's experiment results on ICEWS14_resplit, ICEWS0515, and ICEWS14, respectively. Appendix

B includes the exact metrics.

All models are trained under the same batch size 64 and the same learning rate 0.0001. The models on ICEWS14_resplit and ICEWS0515 are selected at best validation MRR. Based on these premises, the results on ICEWS14_resplit and ICEWS0515 show that:

1. The transformer model's performance is more sensitive to sequence length than RNN, and no monotonic performance change arises as sequence length increases, which contrasts with the common knowledge that transformer could have better performances on longer sequences than RNN due to enlarged prediction capacity [10]. This non-monotonicity, nevertheless, means that one can vary the sequence lengths to obtain a full profile of the model's performances on different lengths.
2. In general, the transformer model is not necessarily better than RNN in terms of metrics, but under certain situations, the transformer's performances are approaching that of RNN's.

The models trained on ICEWS14 cannot be trust-worthily validated, and thus overfitting might occur while not foreseeable. For testing and comparison, we chose models that reach a similar and acceptable low training error. The results on ICEWS14 show that, with the possible existence of overfitting, the transformer model can surpass the model at some lengths. This conclusion is not fully trustable, though, since the results on ICEWS_resplit show a contrary conclusion that the transformer does not surpass RNN, whichever sequence length is under test.

4.4.2 Regarding Model Structure

We tested modifications on several model components and present the effects hereafter. Tested modifications include the total model parameter number, the aggregator scheme, and a certain component in the intensity function.

Parameter Number

The model setting in Section 4.4.1 gives a total parameter number of about 4.9M. As a reference, BERT [24] gives a number of around 110M. We thus tested our model under a boosted parameter space of approximately 23M, which balances our available computational power and model's information capacity. This was done by expanding the transformer's key and value tensor $(\mathbf{K}, \mathbf{V})$ dimensions to 512, rather than 64 previously. We did this hoping that a larger parameter space and a larger hidden state dimension in the encoder would help enrich more information from the sequence.

Table 4.2 shows the results on ICEWS14_resplit and ICEWS0515. We only offer the raw results on minimum and maximum experimented allowed sequence

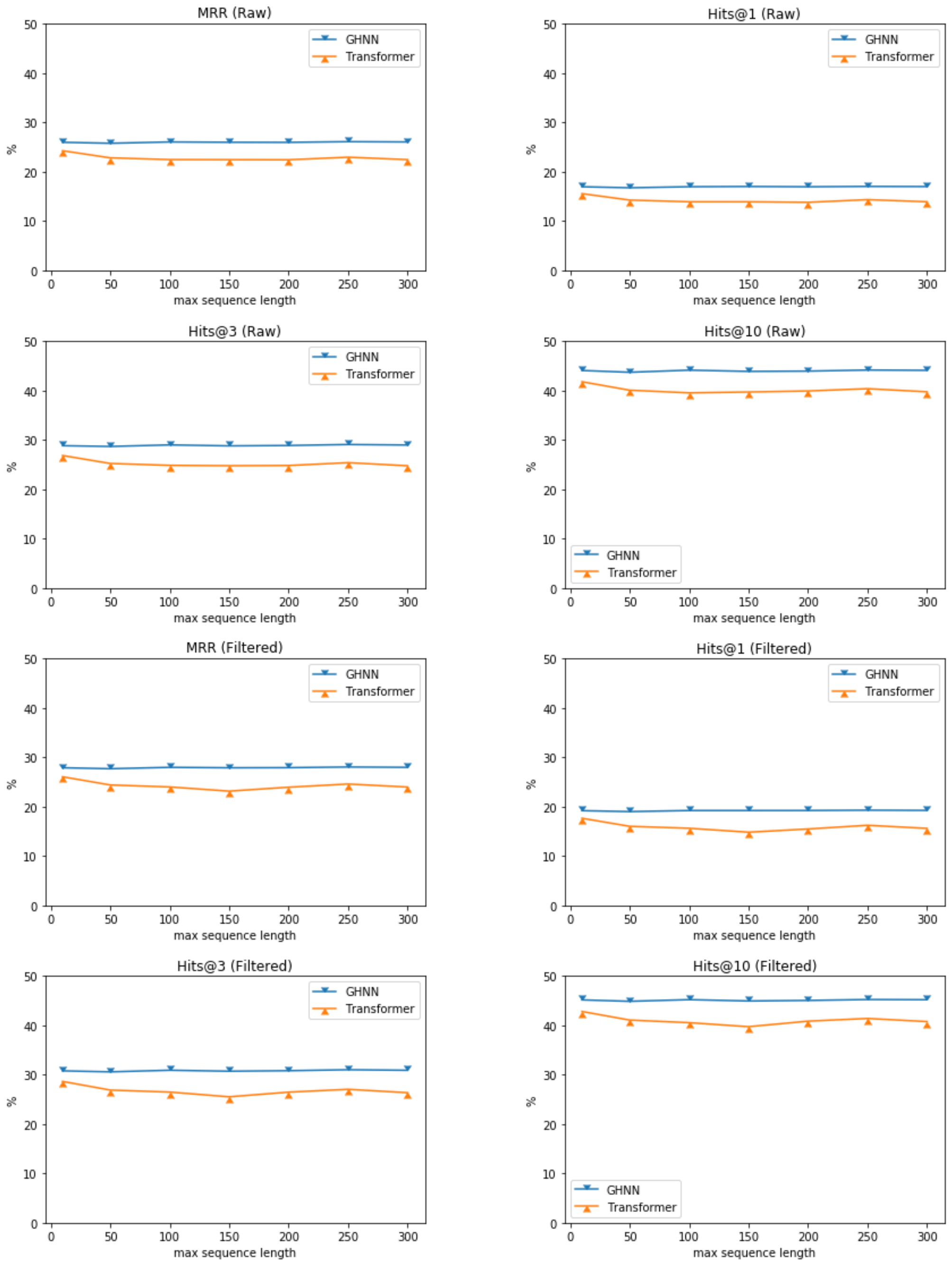


Figure 4.1: Metrics comparison between GHNN and our transformer model, on ICEWS14_resplit dataset

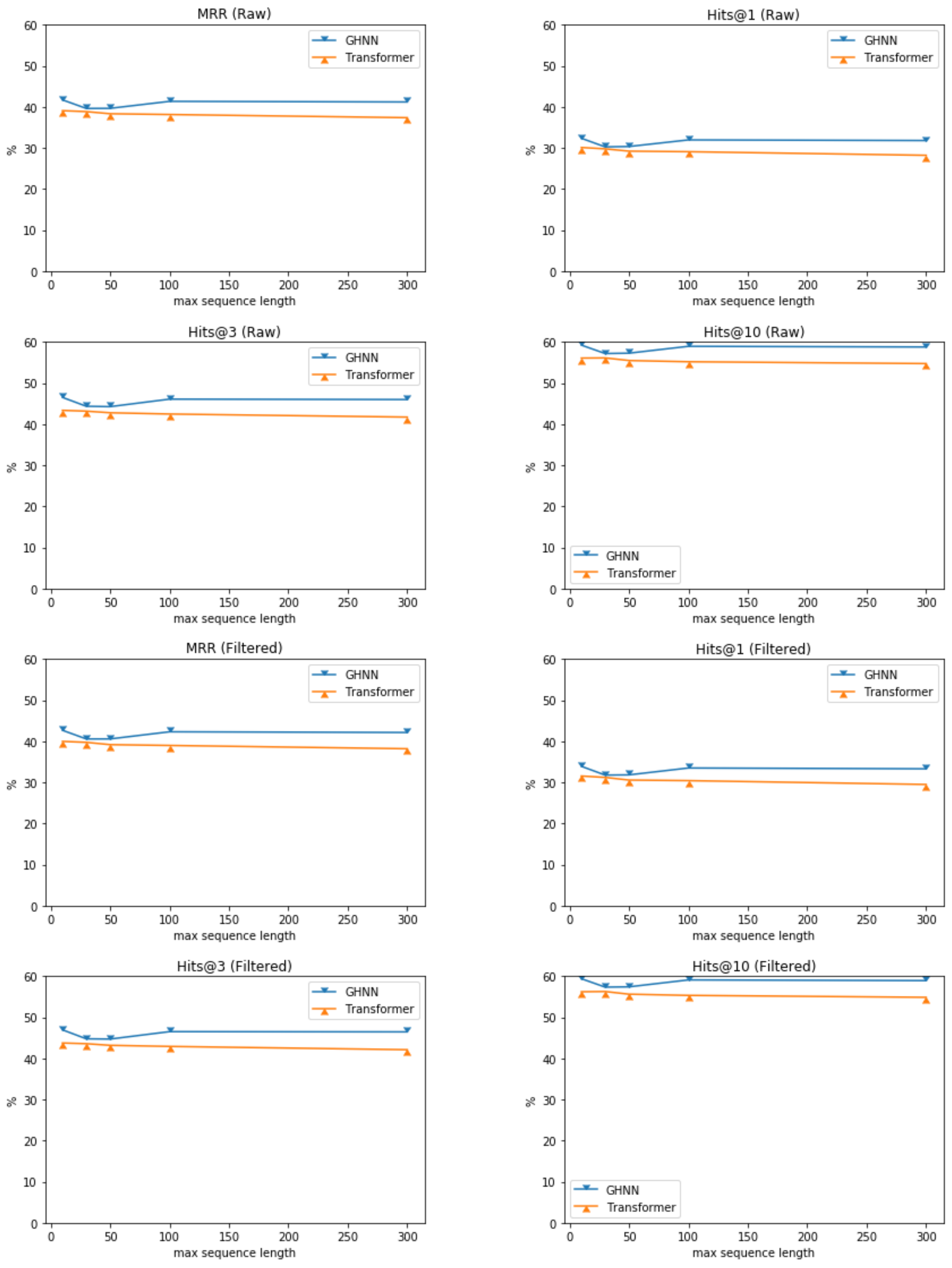


Figure 4.2: Metrics comparison between GHNN and our transformer model, on ICEWS0515 dataset

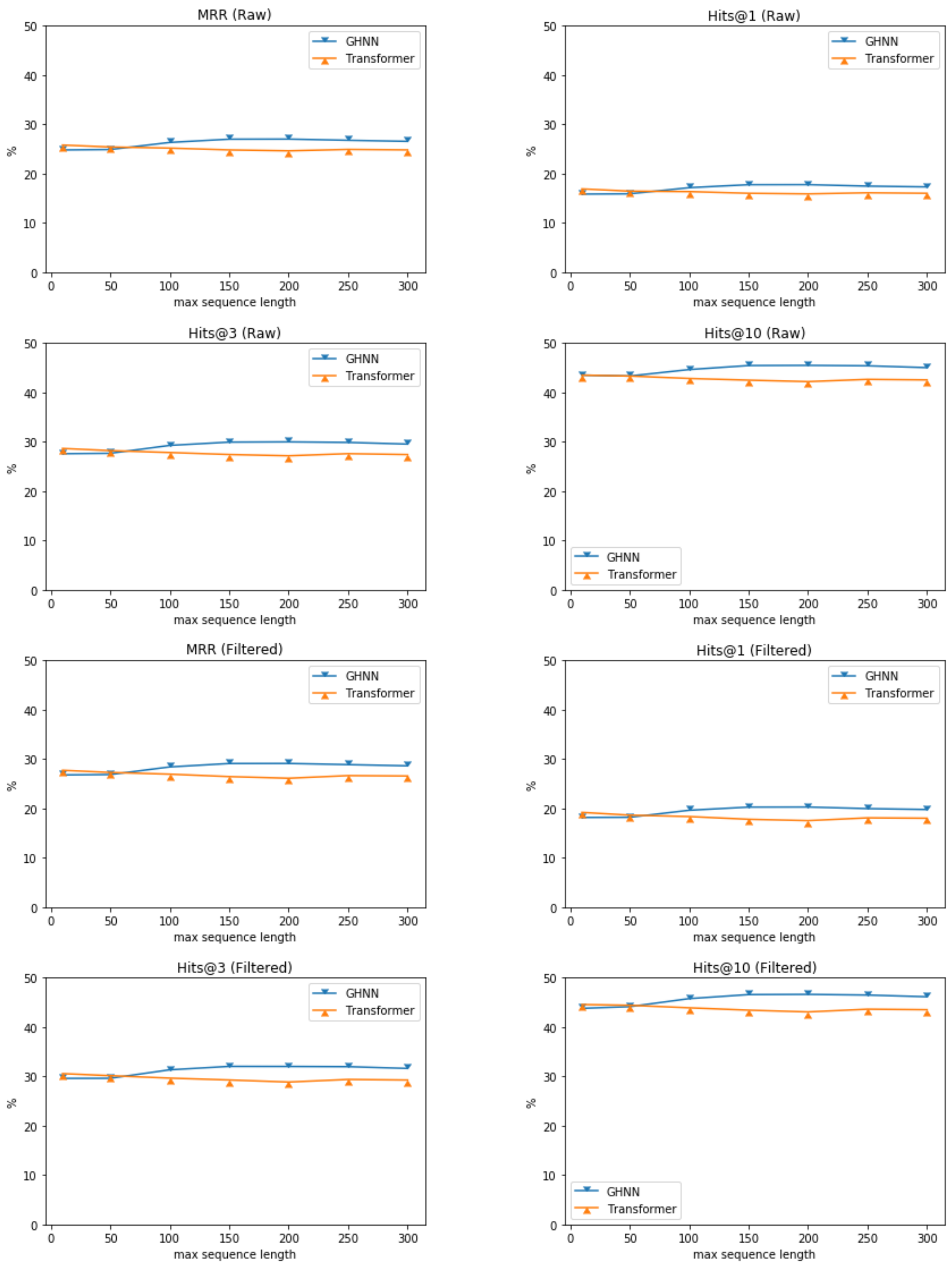


Figure 4.3: Metrics comparison between GHNN and our transformer model, on ICEWS14 dataset

length here since they should be representative, concluded from the previous full length-wise profile and the similar pattern between raw and filtered test results.

$dim(\mathbf{K}), dim(\mathbf{V}) = 512$: no brackets; $(dim(\mathbf{K}), dim(\mathbf{V}) = 64)$: in brackets

Dataset	Max seq length	MRR (%)	Hits@1 (%)	Hits@3 (%)	Hits@10 (%)
ICEWS14_resplit	10	22.9839 (**24.2635**)	14.4027 (**15.5541**)	25.4133 (**26.8560**)	40.3296 (**41.8056**)
	300	21.7073 (22.4572)	13.4199 (13.9082)	23.8055 (24.7748)	38.5812 (39.7678)
ICEWS0515	10	38.2561 (**39.0785**)	29.2283 (**30.1033**)	42.5478 (**43.3634**)	55.3365 (**56.0870**)
	300	34.7373 (37.3733)	25.2679 (28.2037)	38.9229 (41.7242)	52.9257 (54.7506)

Table 4.2: Comparison of the transfomer model under different transformer key and value dimensions.

The results show that enriching the model's parameter number would not help the model better learn the patterns.

Aggregator structure

As introduced in Section 3.2.3, we introduced an additional relational aggregator apart from the mean aggregator. For the relational aggregator, with $\phi(\cdot)$ being the circular-correlation in Eq.3.2, we tested the setting:

$$g\left(\mathbf{O}_t\left(e_{s_i}, e_{r_i}\right)\right) = \frac{1}{\left|\mathbf{O}_t\left(e_{s_i}, e_{r_i}\right)\right|} \sum_{\mathbf{e}_o \in \mathbf{O}_t\left(e_{s_i}, e_{r_i}\right)} w_{r_i} \phi(\mathbf{e}_o, \mathbf{e}_{r_i}), \tag{4.1}$$

where w_{r_i} is from a learnable matrix $\mathbf{W}$ that assigns a weight for each relation type.

Table 4.3 shows the results on ICEWS15. The experiments are conducted under the same batch size, parameter numbers, and learning rate. Again, we only offer results under one sequence length and under the raw test set only, since they should suffice to show the effect.

Dataset: ICEWS0515
Max sequence length: 50

Aggregator	MRR (%)	Hits@1 (%)	Hits@3 (%)	Hits@10 (%)
Mean	**38.3318**	**29.2167**	**42.7792**	**55.4912**
Relational	32.3110	22.9489	36.3132	50.5452

Table 4.3: Comparison of the transformer model under different aggregator structure.

It is evident that the newly introduced compositional aggregator is harmful to the performances.

One possible reason is that the aggregator is imposed onto the group of events that have the same subject/object and relation, and thus introducing operations on the relation embedding ($\mathbf{e}_r$) would compress redundant information into the aggregated tensor, and therefore make the tensor less "informative." One possible improving approach is to group events by the same subject/object and allow for more relation types. Section 5.2 will discuss on more aggregator options.

Temporal encoding

The positional encoding is an essential part of the attention mechanism to make use of sequence orders. On the temporal knowledge graph, the positional encoding idea is implemented by a temporal encoding, which fuses the event time information, as in Eq.2.12.

The Hawkes process assumption requires implementing the temporal encoding with time intervals between events, i.e., t_j is the time difference of the event j from its previous event. This implementation contradicts the original positional encoding, which computes the encoding with the absolute position in a sentence. We, therefore, tested a temporal encoding with the absolute event time.

We also tested the model performance without a temporal encoding to verify its actual contribution to our model. Table 4.4 presents the result.

Dataset: ICEWS14
Max sequence length: 10

Temporal encoding	MRR (%)	Hits@1 (%)	Hits@3 (%)	Hits@10 (%)
Without	24.3396	16.3542	26.7169	40.0044
With, Time interval	**25.8205**	**17.5004**	**28.2841**	**42.3075**
With, Absolute time	19.4451	12.3150	21.0346	33.4834

Table 4.4: Comparison of the transformer model with different temporal encoding scheme.

It is proved that a model without temporal encoding would have degraded performances. Therefore, the temporal encoding is not removable. Also, interestingly, the replacement of time interval to absolute event time would damage the performances profoundly.

Intensity function components

We adjusted the time-related term in the intensity function Eq.3.3, aiming at making the intensity tensor more informative. We experimented on 1) the time term α_t's weight and 2) whether to keep the denominator t_i in the time term.

The result is shown in Table 4.5. Experiments on the same dataset are done under the same batch size, learning rate, and other model specifications.

Dataset: ICEWS0515
Max sequence length: 50
Test method: Raw

Time term	MRR (%)	Hits@1 (%)	Hits@3 (%)	Hits@10 (%)
Without time term ($\alpha_t = 0$)	**38.3318**	**29.2167**	**42.7792**	**55.4912**
$\alpha_t = 0.0001$	37.5861	28.4011	42.0134	54.9120
$\alpha_t = 0.0001$, no denominator ($\alpha_t(t_q - t_i)$)	37.7580	28.6368	42.1544	54.9756

Table 4.5: Comparison of the transformer model under different intensity function components: time term equation and weights.

It is observed the existence/absence of the time term $\alpha_t \frac{t_q - t_i}{t_i}$ does not make a significant difference to the performances.

4.4.3 Regarding Training Hyper-parameters

Some of the hyperparameters during training would cause differences in model performances as well. This section will present such effects from dropout, batch size, and the temperature of the transformer's attention module.

Dropout

Dropout [25] was invented to avoid overfitting of neural networks by preventing feature co-adapting. Baldi et al. [26] prove that a dropout in effect is equivalent to regularization, and a dropout probability of 0.5 imposes the highest level of regularization.

We experimented on our model with dropout probability $p = 0.1$ and $p = 0.5$. Table **??** shows the results.

Dataset: ICEWS14_resplit
Max sequence length: 250
Test method: Raw

Dropout	MRR (%)	Hits@1 (%)	Hits@3 (%)	Hits@10 (%)
$p = 0.1$	**21.5661**	**13.2399**	**23.6500**	**38.4406**
$p = 0.5$	20.9680	12.8873	22.9273	37.2798

Table 4.6: Comparison of the transformer model under different dropout probabilities.

The results indicate that increasing the dropout for a higher level of regularizing effect would slightly degrade the performances. But at the same time, metric curves during training show that increasing dropout ratio speeds up the model's convergence. With $p = 0.1$, the model needed more than 60 epochs of training to converge, while with $p = 0.5$, it only required 30 epochs.

Batch size

It was observed that the model is batch size-sensitive. Here-under, Table 4.7 contains the results from parallel experiments on batch size. We applied gradient accumulation to adjust the effective batch sizes.

Dataset: ICEWS0515
Test method: Raw

Max sequence length	Batch size	MRR (%)	Hits@1 (%)	Hits@3 (%)	Hits@10 (%)
10	64	**38.5386**	**29.3512**	**42.9447**	**56.0371**
	256	38.2561	29.2283	42.5478	55.3365
300	64	**35.4433**	**25.8616**	**39.9005**	**53.6842**
	256	34.7373	25.2679	38.9229	52.9257

Table 4.7: Comparison of the transformer model under different batch sizes.

It is observable that an increase in batch size would lead to a slight decrease in performance. We only show the raw test results on some lengths, and results from the remaining settings show the same phenomenon.

Transformer attention temperature

The transformer attention temperature $\sqrt{M_K}$ in Eq.2.13 controls the softness of the attention score distribution [27] by controlling the softmax function's smoothness. The higher the value, the smaller the attention score differences between more and less "attended" classes.

Table 4.8 presents the results. The models are trained with the same model specifications and for a similar epoch number.

Results show that the temperatures of $(M_K)^{0.3}$ and $(M_K)^{0.8}$ perform similarly better than the other two, while $\sqrt{M_K}$ is usually the default choice [28]. We still set the power to be 0.5 for all other experiments since the improvement from other values is rather trivial.

Dataset: ICEWS14
Max sequence length: 10
Test method: Raw

Temperature	MRR (%)	Hits@1 (%)	Hits@3 (%)	Hits@10 (%)
$(\mathrm{M}_K)^{0.3}$	**26.0919**	**17.6489**	28.6415	**42.8802**
$\sqrt{M_K}$	25.8205	17.5004	28.2841	42.3075
$(\mathrm{M}_K)^{0.8}$	26.0719	17.6326	**28.6496**	42.8508
M_K	25.8691	17.4572	28.5060	42.4462

Table 4.8: Comparison of the transformer model under different temperatures.

Chapter 5

Discussion and Future Works

On our implementation of the transformer in the temporal knowledge graph, we have observed an approaching yet not beating performance than RNN in past works. We have also observed modules that bring slight improvements at best, including a different aggregator scheme, the transformer's structure, the intensity function, some training hyperparameters, etc. It is plausible, with all these experiment results, that the improvement, if any, should come from other modules, i.e., event embedding, more complex aggregators, other schemes for temporal encoding, and so on. Following are some works and ideas that might be helpful for our task.

5.1 Another Sequence Generating Logic

The current event sequence generation scheme is by filtering the events with the same subject/object and relation. We could expand the grouped event pool to only restricting one same entity. More events will be passed to the aggregator, and potentially longer sequences can be generated.

5.2 More Complicated Aggregators

If with the new sequence generating logic above, the current mean and relational aggregator naturally becomes more complicated because more events are aggregated at each timestamp. Additionally, the relation-dependent weight in the relational aggregator can be more effective on the new sequence because different relations exist in events at the same timestamp, and thus more relation-dependent features can be potentially "extracted."

A similar idea that imposes different weights to events at one timestamp is attention-based aggregation [29], such that the assignment of aggregator's weight includes the attention mechanism already.

Another direction to introduce more events to the aggregator, even though some events are not directly/strongly related to the queried one. In a graph structure, this can be realized by a multi-hop neighborhood aggregation [19], i.e. for the interested event (s, r, o, t), not only its direct (one-hop) neighbors with the same $(s, r, o\prime, t)$ will be aggregated, but also its two-hop neighbors $(s\prime\prime, r\prime\prime, o\prime, t)$, and so on. Some restrictions on aggregated events can also help shrink the aggregation pool size, e.g., allowing two hops at maximum or allowing only multi-hop neighbors with the same r.

5.3 Informer

Informer [10] is a newly proposed transformer-based model designed explicitly for long sequence time-series forecasting. It claims to achieve a higher computing efficiency and better predicting performances, with self-attention distilling layers. It is possible and plausible to replace the transformer with the informer, especially on longer sequence lengths.

5.4 Another Temporal Encoding Scheme

Also inspired by Informer [10], the temporal encoding can be replaced by a summation of more decomposed embeddings, including position embedding and week or month embeddings. By such decomposition, the model can take into consideration both relative time interval and absolute event time as input simultaneously.

Chapter 6

Conclusion

In this thesis work, we proposed and implemented a transformer-based model for event forecasting on the temporal knowledge graph. The model also contains modules of mean aggregator and predictor based on Hawkes process.

We conducted experiments with the model on several widely used temporal knowledge graph datasets. We compared with the baseline model, GHNN, an LSTM-based model on the same task. The model has achieved approaching results compared to GHNN.

We also conducted experiments on several individual modules in the model and some hyperparameters during training. Results show that the model reaches its best performances with relatively small batch size and parameter number, a slight deviation from the typical attention temperature, a relatively small dropout ratio, and an intensity function that contains no explicit time information. Other modifications on aggregator structure and temporal encoding have proved to be counter-effective.

For further potential improvements, one can turn to new event sequence generation schemes, other aggregator options, different temporal encodings, or even alternatives for the transformer.

Bibliography

[1] A. G. Hawkes, "Spectra of some self-exciting and mutually exciting point processes," *Biometrika*, vol. 58, no. 1, pp. 83–90, 1971. [Online]. Available: http://www.jstor.org/stable/2334319

[2] H. Mei and J. Eisner, "The neural hawkes process: A neurally self-modulating multivariate point process," *CoRR*, vol. abs/1612.09328, 2016. [Online]. Available: http://arxiv.org/abs/1612.09328

[3] S. Zuo, H. Jiang, Z. Li, T. Zhao, and H. Zha, "Transformer hawkes process," *arXiv preprint arXiv:2002.09291*, 2020.

[4] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko, "Translating embeddings for modeling multi-relational data," in *Neural Information Processing Systems (NIPS)*, 2013, pp. 1–9.

[5] T. Trouillon, J. Welbl, S. Riedel, É. Gaussier, and G. Bouchard, "Complex embeddings for simple link prediction," in *International Conference on Machine Learning*. PMLR, 2016, pp. 2071–2080.

[6] Z. Sun, Z. Deng, J. Nie, and J. Tang, "Rotate: Knowledge graph embedding by relational rotation in complex space," *CoRR*, vol. abs/1902.10197, 2019. [Online]. Available: http://arxiv.org/abs/1902.10197

[7] Z. Han, Y. Ma, Y. Wang, S. Günnemann, and V. Tresp, "Graph hawkes neural network for forecasting on temporal knowledge graphs," 2020.

[8] K. Leetaru and P. A. Schrodt, "Gdelt: Global data on events, location, and tone," *ISA Annual Convention*, 2013.

[9] E. Boschee, J. Lautenschlager, S. O'Brien, S. Shellman, J. Starz, and M. Ward, "ICEWS Coded Event Data," 2015. [Online]. Available: https://doi.org/10.7910/DVN/28075

[10] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang, "Informer: Beyond efficient transformer for long sequence time-series forecasting," 2020.

[11] S. Karita, N. Chen, T. Hayashi, T. Hori, H. Inaguma, Z. Jiang, M. Someki, N. E. Y. Soplin, R. Yamamoto, X. Wang, S. Watanabe, T. Yoshimura, and W. Zhang, "A comparative study on transformer vs rnn in speech applications," 2019.

[12] C. Palm, *Intensitätsschwankungen im Fernsprechverkehr*, ser. Ericsson technics. L. M. Ericcson, 1943. [Online]. Available: https://books.google.ch/books?id=5cy2NQAACAAJ

[13] P. J. Laub, T. Taimre, and P. K. Pollett, "Hawkes processes," *arXiv preprint arXiv:1507.02822*, 2015.

[14] M.-A. Rizoiu, Y. Lee, S. Mishra, and L. Xie, "A tutorial on hawkes processes for events in social media," *arXiv preprint arXiv:1708.06401*, 2017.

[15] Ontotext. What is a knowledge graph? [Online]. Available: https://www.ontotext.com/knowledgehub/fundamentals/what-is-a-knowledge-graph/

[16] A. García-Durán, S. Dumancic, and M. Niepert, "Learning sequence encoders for temporal knowledge graph completion," *CoRR*, vol. abs/1809.03202, 2018. [Online]. Available: http://arxiv.org/abs/1809.03202

[17] B. Yang, W. Yih, X. He, J. Gao, and L. Deng, "Learning multi-relational semantics using neural-embedding models," *CoRR*, vol. abs/1411.4072, 2014. [Online]. Available: http://arxiv.org/abs/1411.4072

[18] S. Vashishth, S. Sanyal, V. Nitin, and P. Talukdar, "Composition-based multi-relational graph convolutional networks," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=BylA_C4tPr

[19] Z. Sun, C. Wang, W. Hu, M. Chen, J. Dai, W. Zhang, and Y. Qu, "Knowledge graph alignment network with gated multi-hop neighborhood aggregation," *CoRR*, vol. abs/1911.08936, 2019. [Online]. Available: http://arxiv.org/abs/1911.08936

[20] Y. Zhou, S. Shi, and H. Huang, "Weighted aggregator for the open-world knowledge graph completion," in *Data Science*, J. Zeng, W. Jing, X. Song, and Z. Lu, Eds. Singapore: Springer Singapore, 2020, pp. 283–291.

[21] Q. Wang, Z. Mao, B. Wang, and L. Guo, "Knowledge graph embedding: A survey of approaches and applications," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 12, pp. 2724–2743, 2017.

[22] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[23] albanD. (2017) Why do we need to set the gradients manually to zero in pytorch? [Online]. Available: https://discuss.pytorch.org/t/why-do-we-need-to-set-the-gradients-manually-to-zero-in-pytorch/4903/20?u=alband

[24] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” *arXiv preprint arXiv:1810.04805*, 2018.

[25] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[26] P. Baldi and P. J. Sadowski, “Understanding dropout,” in *Advances in Neural Information Processing Systems*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds., vol. 26. Curran Associates, Inc., 2013. [Online]. Available: https://proceedings.neurips.cc/paper/2013/file/71f6278d140af599e06ad9bf1ba03cb0-Paper.pdf

[27] J. Lin, X. Sun, X. Ren, M. Li, and Q. Su, “Learning when to concentrate or divert attention: Self-adaptive attention temperature for neural machine translation,” in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, Oct.-Nov. 2018, pp. 2985–2990. [Online]. Available: https://www.aclweb.org/anthology/D18-1331

[28] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” 2017.

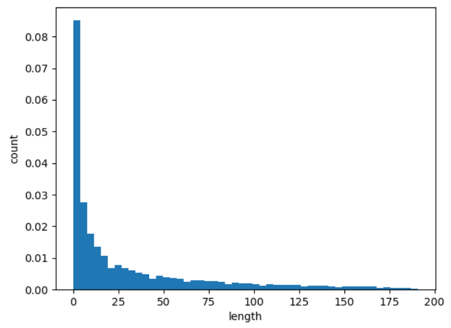
[29] M. Zhao, W. Jia, and Y. Huang, “Attention-based aggregation graph networks for knowledge graph information transfer,” in *Advances in Knowledge Discovery and Data Mining*, H. W. Lauw, R. C.-W. Wong, A. Ntoulas, E.-P. Lim, S.-K. Ng, and S. J. Pan, Eds. Cham: Springer International Publishing, 2020, pp. 542–554.

Appendix A

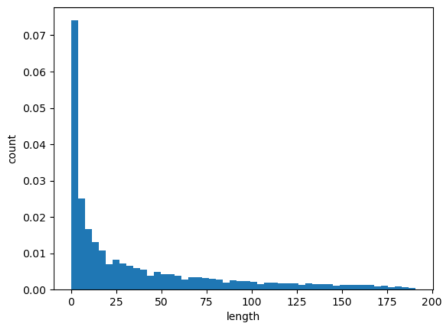
Statistics of Generated Event Sequences

TableA.1 - A.3 are statistics of the generated sequences.

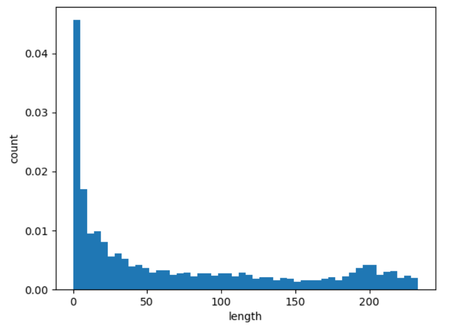
Dataset: ICEWS14_resplit



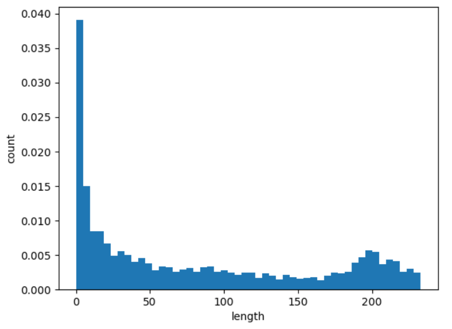
Sub-centric, Train
Avg length: 31.8
Max length: 191



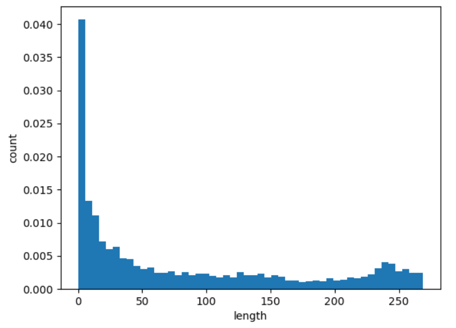
Obj-centric, Train
Avg length: 37.0
Max length: 191



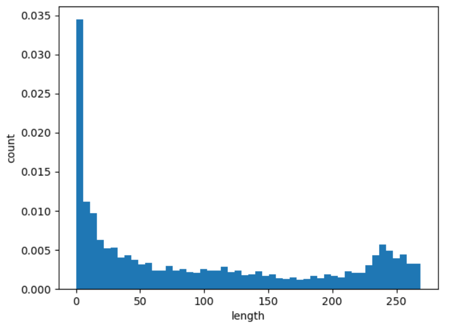
Sub-centric, Valid
Avg length: 70.7
Max length: 233



Obj-centric, Valid
Avg length: 81.8
Max length: 233



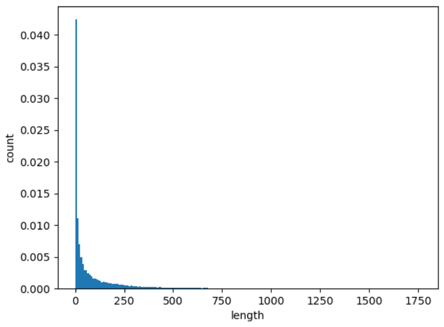
Sub-centric, Test
Avg length: 82.4
Max length: 269



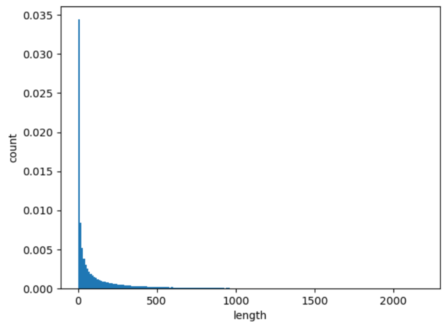
Obj-centric, Test
Avg length: 97.2
Max length: 269

Table A.1: Statistics of generated sequences from ICEWS14_resplit

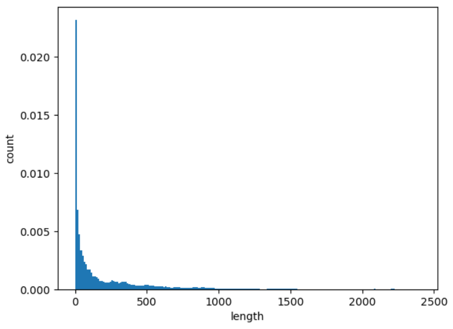
Dataset: ICEWS0515



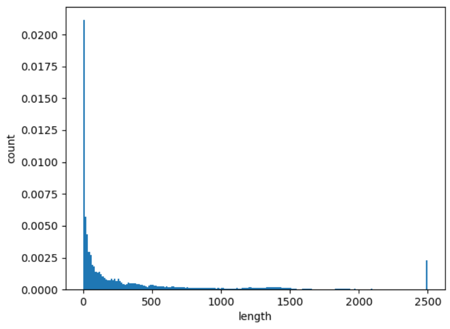
Sub-centric, Train
Avg length: 101.4
Max length: 1762



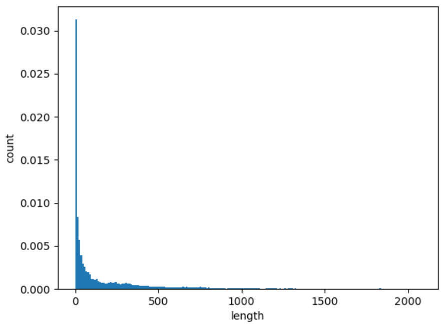
Obj-centric, Train
Avg length: 145.4
Max length: 2186



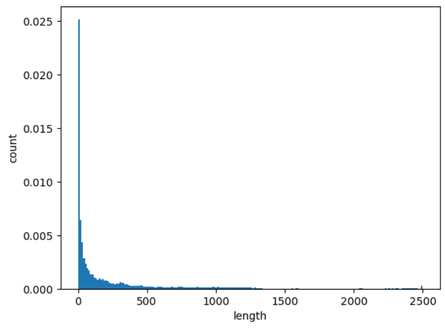
Sub-centric, Valid
Avg length: 243.9
Max length: 2405



Obj-centric, Valid
Avg length: 363.17
Max length: 2500



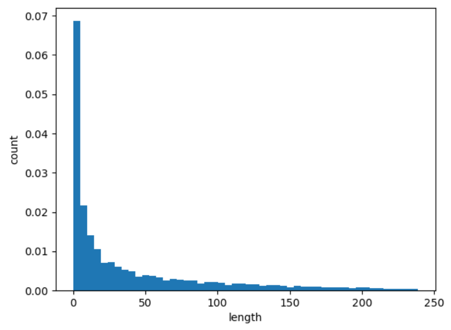
Sub-centric, Test
Avg length: 190.2
Max length: 2077



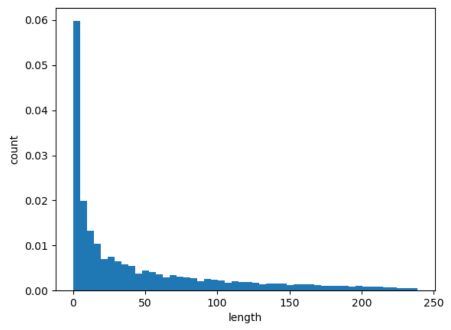
Obj-centric, Test
Avg length: 290.5
Max length: 2500

Table A.2: Statistics of generated sequences from ICEWS0515

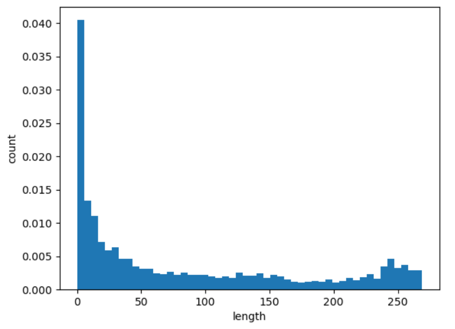
Dataset: ICEWS14



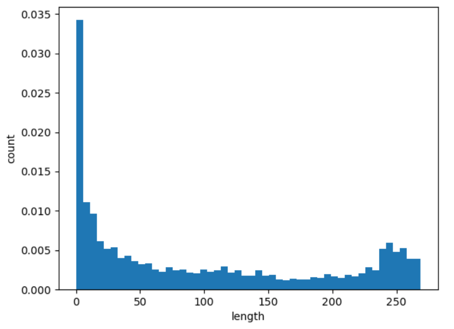
Sub-centric, Train
Avg length: 39.8
Max length: 239



Obj-centric, Train
Avg length: 46.2
Max length: 239



Sub-centric, Test
Avg length: 83.3
Max length: 269



Obj-centric, Test
Avg length: 98.2
Max length: 269

Table A.3: Statistics of generated sequences from ICEWS14

Appendix B

Experiment Results of Transformer and GHNN

Table B.1, B.2 and B.3 show the test metrics of our transformer model and GHNN verbatim. The metrics of the transformer model is in the first line of each cell, without a bracket. The metrics of GHNN are in the second line of each cell and bracketed.

Dataset: ICEWS14_resplit
Transformer: no brackets; (GHNN) in brackets

Test Method	Max seq length	MRR (%)	Hits@1 (%)	Hits@3 (%)	Hits@10 (%)
Raw	10	**24.2635** (25.9753)	**15.5541** (16.9506)	**26.8560** (28.8456)	**41.8056** (44.0858)
	50	22.8221 (25.7760)	14.2506 (16.7414)	25.2476 (28.6927)	40.0952 (43.7258)
	100	22.4640 (26.0422)	13.9062 (16.9547)	24.8604 (29.0038)	39.5559 (44.1530)
	150	22.4465 (25.9803)	13.9035 (17.0022)	24.7850 (28.8129)	39.7291 (43.9038)
	200	22.4297 (25.9620)	13.7941 (16.9458)	24.8305 (28.8883)	39.9370 (43.9608)
	250	22.9480 **(26.1005)**	14.3369 **(17.0178)**	25.4065 **(29.0867)**	40.3982 **(44.1775)**
	300	22.4572 (26.0428)	13.9082 (16.9880)	24.7748 (28.9821)	39.7678 (44.1170)
Filtered	10	**26.0654** (27.8903)	**17.6624** (19.1982)	**28.6418** (30.7794)	**42.8136** (45.1610)
	50	24.4085 (27.6936)	16.0275 (19.0067)	26.8900 (30.5797)	41.0747 (44.8737)
	100	24.0040 (27.9783)	15.6308 (19.2274)	26.4831 (30.9077)	40.5496 (45.2106)
	150	23.1554 (27.8883)	14.8341 (19.2261)	25.5247 (30.7060)	39.7325 (44.9417)
	200	23.9509 (27.9113)	15.4848 (19.2335)	26.4689 (30.8127)	40.8553 (45.0510)
	250	24.5991 **(28.0511)**	16.2340 **(19.2974)**	27.0428 **(31.0076)**	41.3967 **(45.2344)**
	300	23.9865 (27.9802)	15.6091 (19.2553)	26.3582 (30.8969)	40.7684 (45.1984)

Table B.1: Comparison of our model and GHNN on ICEWS14_resplit under different maximum sequence lengths, with same batch size and learning rate

Dataset: ICEWS0515

Transformer: no brackets; (GHNN) in brackets

Test Method	Max seq length	MRR (%)	Hits@1 (%)	Hits@3 (%)	Hits@10 (%)
Raw	10	**39.0785** **(41.7107)**	**30.1033** **(32.3514)**	**43.3634** **(46.5168)**	56.0870 **(59.2419)**
	30	38.8496 (39.6275)	29.7728 (30.2537)	43.1725 (44.3591)	**56.1261** (57.1955)
	50	38.3318 (39.6500)	29.2167 (30.3469)	42.7792 (44.2666)	55.4912 (57.2809)
	100	38.1439 (41.3510)	29.0837 (31.9537)	42.4711 (46.0931)	55.1947 (58.9512)
	300	37.3733 (41.2009)	28.2037 (31.8040)	41.7242 (45.9962)	54.7506 (58.7878)
Filtered	10	**39.9818** **(42.6792)**	**31.5509** **(33.9096)**	**43.7517** **(46.9558)**	56.2273 **(59.3981)**
	30	39.7459 (40.5724)	31.2096 (31.7772)	43.5832 (44.7706)	**56.2815** (57.3510)
	50	39.1854 (40.5790)	30.5675 (31.8322)	43.1863 (44.7113)	55.6329 (57.4219)
	100	38.9896 (42.3157)	30.4272 (33.5004)	42.9209 (46.5168)	55.3415 (59.1031)
	300	38.1888 (42.1473)	29.4980 (33.3030)	42.1190 (46.4597)	54.8693 (58.9368)

Table B.2: Comparison of our model and GHNN on ICEWS0515 under different maximum sequence lengths, with same batch size and learning rate

Dataset: ICEWS14
Transformer: no brackets; (GHNN) in brackets

Test Method	Max seq length	MRR (%)	Hits@1 (%)	Hits@3 (%)	Hits@10 (%)
Raw	10	**25.7936** (24.8037)	**16.8910** (15.8443)	**28.6570** (27.5866)	**43.5133** (42.7554)
	50	25.4007 (24.8824)	16.4382 (15.8908)	28.2148 (27.6682)	43.3322 (43.0393)
	100	25.1747 (26.3252)	16.3436 (17.1407)	27.8273 (29.2680)	42.8402 (44.6395)
	150	24.8077 (26.9833)	16.0067 (17.7468)	27.4218 (29.9370)	42.4943 (45.4558)
	200	24.6169 (**27.0031**)	15.8786 (**17.7591**)	27.1664 (**29.9835**)	42.1990 (**45.4909**)
	250	24.9073 (26.7576)	16.1103 (17.4523)	27.5972 (29.8758)	42.6583 (45.4060)
	300	24.8124 (26.5435)	16.0034 (17.3022)	27.4104 (29.5373)	42.5425 (45.0332)
Filtered	10	**27.7326** (26.7953)	**19.1949** (18.1702)	**30.5571** (29.5952)	**44.5625** (43.7964)
	50	27.2926 (26.8653)	18.6426 (18.2029)	30.1467 (29.6131)	44.4132 (44.1268)
	100	26.9454 (28.4106)	18.3522 (19.6281)	29.6474 (31.3321)	43.8984 (45.7577)
	150	26.4500 (29.1052)	17.8015 (20.2686)	29.2713 (**32.0272**)	43.4219 (46.5784)
	200	26.1179 (**29.1167**)	17.5249 (**20.2849**)	28.8528 (32.0117)	43.0687 (**46.6208**)
	250	26.6493 (28.8811)	18.0911 (19.9626)	29.3782 (31.9693)	43.6210 (46.4838)
	300	26.5730 (28.6337)	18.0152 (19.7807)	29.2607 (31.6127)	43.5158 (46.1395)

Table B.3: Comparison of our model and GHNN on ICEWS14 under different maximum sequence lengths, with same batch size and learning rate. Both performances are measured with a model that has an similarly low train error, but not necessarily right before overfitting.