SEQUENCE LEARNING WITH SIDE INFORMATION: MODELING AND APPLICATIONS

By

Zhiwei Wang

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

Computer Science — Doctor of Philosophy

2020

ABSTRACT

SEQUENCE LEARNING WITH SIDE INFORMATION: MODELING AND APPLICATIONS

By

Zhiwei Wang

Sequential data is ubiquitous and modeling sequential data has been one of the most long-standing computer science problems. The goal of sequence modeling is to represent a sequence with a low-dimensional dense vector that incorporates as much information as possible. A fundamental type of information contained in sequences is the sequential dependency and a large body of research has been devoted to designing effective ways to capture it. Recently, sequence learning models such as recurrent neural networks (RNNs), temporal convolutional networks, and Transformer have gained tremendous popularity in modeling sequential data. Equipped with effective structures such as gating mechanisms, large receptive fields, and attention mechanisms, these models have achieved great success in many applications of a wide range of fields.

However, besides the sequential dependency, sequences also exhibit side information that remains under-explored. Thus, in the thesis, we study the problem of sequence learning with side information. Specifically, we present our efforts devoted to building sequence learning models to effectively and efficiently capture side information that is commonly seen in sequential data. In addition, we show that side information can play an important role in sequence learning tasks as it can provide rich information that is complementary to the sequential dependency. More importantly, we apply our proposed models in various real-world applications and have achieved promising results.

ACKNOWLEDGMENTS

I have enjoyed an invaluable and memorable experience in the past five years as a Ph.D. student at Michigan State University. From an undergraduate student who barely knew anything about machine learning and data mining, I have grown as a fruitful independent researcher in this field. I deeply understand that such achievement could not be possible without the help, support, and guidance from many people along the way.

First and foremost, my deepest thanks go to my advisor Dr. Jiliang Tang. Without his support, I would not have the opportunity to do research in machine learning. As an advisor, he always offers help, advice, encouragement, and inspiration to me. As a researcher, he greatly influenced me with his passion, perseverance, curiosity, and creativity. As a role model for life, he teaches me through his words and actions the value of kindness, optimism, ambition, and responsibility. I am really grateful to him.

I would like to thank Dr. Arun A. Ross, Dr. Pang-Ning Tan, Dr. Yuying Xie, and Dr. Zitao Liu, for being on my thesis committee. I also want to express my gratitude to Dr. Kevin Liu, who has introduced me to the research world. In addition, I feel fortunate to be a member of the Data Science and Engineering (DSE) lab at Michigan State University. It has been an unforgettable experience working with those amazing labmates, that is, Dr. Tyler Derr, Yao Ma, Xiangyu Zhao, Xiaorui Liu, Hamid Karimi, Haochen Liu, Han Xu, Jamell Dacon, Wentao Wang, Wei Jin, Yaxin Li, Yiqi Wang, Juanhui Li, and Harry Shomer.

Moreover, I appreciate the experience of working with the extraordinary collaborators I met during internships. They are Dr. Dawei Yin, Dr. Yulong Gu, Dr. Zitao Liu, Wenbiao Ding, Hang Li, Tianqiao Liu, Dr. Zhengzhang Chen.

Lastly, I would like to thank my parents and cousin for their love and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

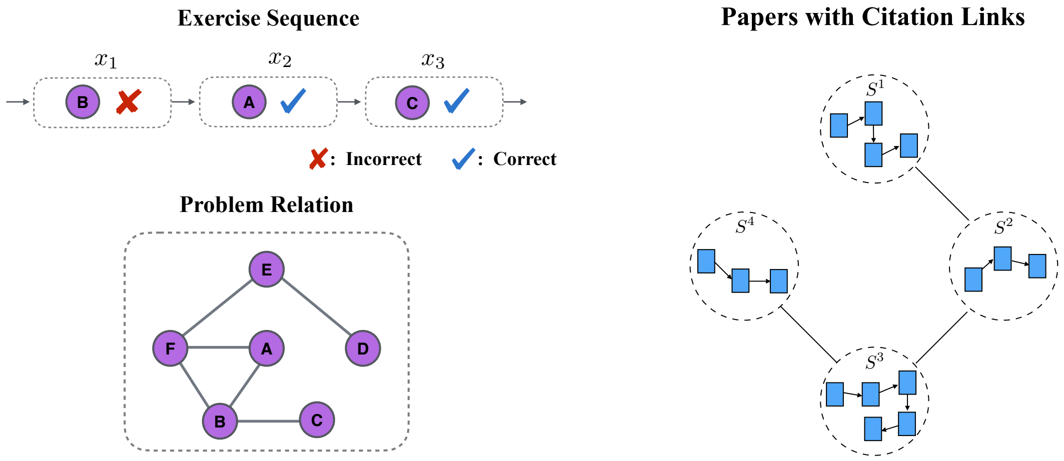
Chapter 1

Introduction

1.1 Motivation

Sequential data, which consists of a set of sequences presented by lists of events with particular orders, are ubiquitous. For example, in genetics, a gene which is a sequence of four different types of nucleotide is sequential data; in linguistics, sentences that are formed by words from a certain vocabulary are considered as sequential data; in e-commerce, the behaviors of a customer in a session are sequential data. Given its wide applications, modeling sequential data has been one of the most long-standing computer science problems.

The goal of sequence modeling is to represent a sequence with a low-dimensional dense vector that incorporates as much information as possible. A fundamental type of information contained in sequences is the sequential dependency of events and a large body of research has been devoted to capturing them. Recently, deep neural networks has gained great attention for modeling sequential data. In particular, recurrent neural networks(RNNs) [101], temporal convolutional networks [10, 26], and Transformer [117] are among the most popular neutral sequence models. Equipped with gating mechanisms, large receptive field, and attention mechanisms, these models are very effective in capturing long-term dependencies in sequences and have led to great success in many applications of a wide range of fields, including linguistics, e-commerce, and education [26, 56, 97].



(a) Student Exercise Sequence.

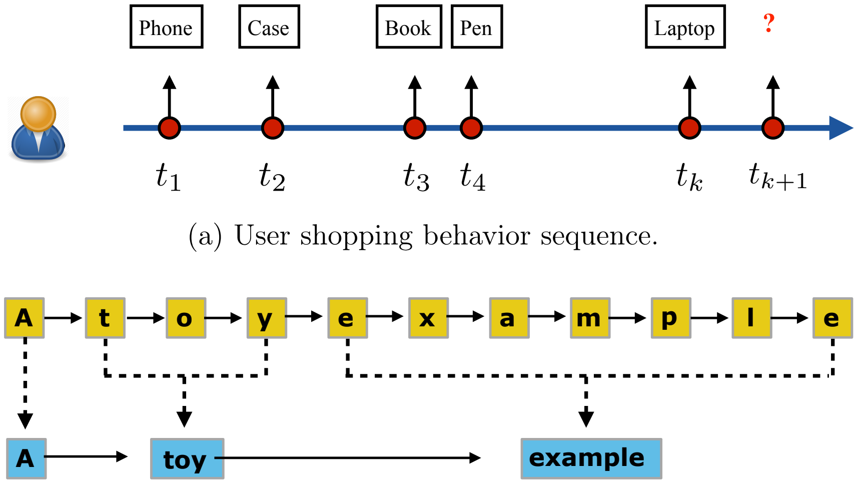
(b) Four papers with their citation links.

Figure 1.1: Examples of event and sequence relation information

However, besides long-term sequential dependency, sequences also exist side information that remains under-explored. In particular, this dissertation focuses on the following five types of side information that are commonly seen.

- Event relation. Most of the existing popular sequence models assume that events are independent. However, this assumption is not true and events are related in many scenarios. Figure 1.1a shows an example of student exercise sequence, where an event involves one problem the student solves. In this figure, problems are inherently related because of the underlying knowledge or skills that are required. Such problem relation provides us essential information to model the student knowledge state [19].

- Sequence relation. Besides event relations, sequences can also be correlated. For instance, figure 1.1b shows four published papers (denoted by S^1, S^2, S^3, S^4) that are connected by their citation relations. In this case, each paper (S^i) is a sequence of English words. To understand the topic of one paper, both content (sequential dependency) and citations (sequence relation) can play an important role.



(a) User shopping behavior sequence.

(b) A toy English sentence.

Figure 1.2: Examples of temporal and hierarchy information

- Temporal information. Sequences are also associated with temporal information. For example, each event of a user behavior sequence in an e-commerce website is recorded with a timestamp indicating when the behavior happens as shown in the figure 1.2a. In this case, the events no longer evenly distributed along the timeline. Such temporal information enables us to not only model user interest but also its dynamics.

- Hierarchy information. Events can exhibit hierarchical structures. One example is the English language where a sentence consists of a sequence of words and a word consists of a sequence of characters. Figure 1.2b gives a concrete example sentence as well as its hierarchical structures.

- Multi-scale sequential dependency. The sequential dependency of events in a sequence is often of multiple scales. Consider the example of anomalous sequence detection problem shown in figure 1.3, where there are three user behavior sequences and two of them are anomalous. Specifically, in the second sequence, the anomalous user tries to hack an account but failed three times before success. In the third sequence, the

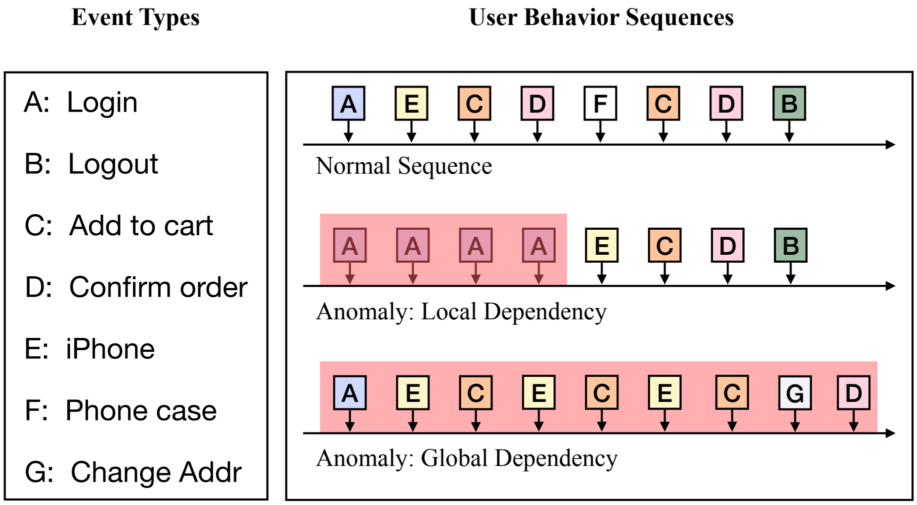


Figure 1.3: Examples of event and sequence relation information

anomalous user adds several expensive identical items and changes the shipping address before checking out. Thus, suspicious behaviors can be reflected by both local (second sequence) and the global (third sequence) sequential dependencies.

In summary, the side information mentioned above can play an essential role in sequence learning tasks as it could provide rich information of sequences that complements the sequential dependency. Thus, in this dissertation, we study the problem of incorporating side information for sequence modeling. In the next section, we give a summary of our contribution.

1.2 Dissertation Contribution

This dissertation presents our efforts to design effective models to capture the side information mentioned above for sequence learning and apply them in various real-world tasks. As a summary, the major contribution of this thesis are listed below.

- We conduct pioneering research to explore event relations for sequence modeling .

We design a neural model that is able to explicitly incorporate the event relation for sequence modeling and empirically verify its effectiveness.

- We are among the first to identify the importance of sequence relation for learning better representations of sequences. We develop an advanced framework that captures the dependencies not only within sequences but also among sequences. We evaluate the framework with extensive experiments and demonstrate the contribution of sequence relation to sequence learning tasks.

- We provide a principled approach to model the temporal dynamics of a sequence and build a novel recommender system that is able to recommend the right item at the right time.

- We tackle the word recognition task where the computer system needs to recognize the correct form of noised words. To achieve it, we design an effective neural sequence model to capture the hierarchical structure of English languages. We believe such a model can serve as a very important component in robust natural language processing systems.

- We design a novel one-class classifier for sequences and it is aware of both global and local scales of sequential dependencies. Moreover, we apply it in sequence anomaly detection problem and show very promising results with comprehensive experiments.

In the next, we will give the outline of this dissertation.

1.3 Dissertation Outline

Based on different types of side information, the rest of this dissertation is organized as follows: Chapter 3 demonstrates our work on capturing event relation for sequence modeling; In Chapter 4, we focus on incorporating relation among sequences; We discuss our work on modeling temporal information in Chapter 5; Chapter 6 introduces our work on modeling hierarchy information in sequences; We present a novel one-class sequence classifier that captures multi-scale dependencies for anomaly detection problem in Chapter 7; Finally, Chapter 8 concludes our works and identifies promising future directions.

Chapter 2

Knowledge Background

In this chapter, we overview the knowledge background of sequence modeling. Specifically, we firstly review sequence modeling techniques, which covers the basics of widely used deep neural networks for sequence modeling. In the second part, we will introduce the typical tasks for sequence modeling as well as their applications.

2.1 Sequence Modeling Techniques

2.1.1 Basic Notations And Definitions

Before we give detailed mathematic formulation of sequence models, we want firstly describe the basic notations and definitions that will be used throughout this chapter. We denote scalars by lower-case letters such as i and j, vectors are denoted by bold lower-case letters such as $\mathbf{x}$ and $\mathbf{h}$, and matrices are represented by bold upper case letters such as $\mathbf{W}$ and $\mathbf{U}$. For a matrix $\mathbf{A}$, we denote the entry at the i^{th} row and j^{th} column of it as $\mathbf{A}(i,j)$, the i^{th} row as $\mathbf{A}(i,:)$ and j^{th} column as $\mathbf{A}(:,j)$. In addition, We use $(\cdots)$ to represent an event sequence and subscripts are used to index the events in the sequence such as (x_1, x_2, x_3).

2.1.2 Recurrent Neural Networks

The most popular deep neural networks are arguably recurrent neural networks(RNNs) [101]. RNNs are fundamentally different to any other neural networks such feedforward networks and convolutional neural networks and specialized for modeling sequential data. Most RNNs process the events one by one according to their sequential order. During this procedure, they maintain an internal state to encode the information that has already been processed. Due to this specially designed process, they are able to scale to sequences of various length. Concretely, let's denote the internal state and the representation of event at step t as $\mathbf{h}_t$ and $\mathbf{x}_t$, respectively. To obtain the state, a vanilla RNN takes the previous state $\mathbf{h}_{t-1}$ and $\mathbf{x}_t$ as input and processes them with a neural cell defined as follows:

$$\mathbf{h}_t = f(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t) \tag{2.1}$$

Where $\mathbf{U}$ and $\mathbf{W}$ are the trainable parameters and $f(\cdot)$ is a activation function which enables the non-linearity. The above described vanilla RNN naturally encodes the historical information and enjoys many advantages. For example, the model size will not changes with the size of the input sequence. However, there are two major drawbacks. One is that it suffers from gradients vanishing or exploding issues, which fail the learning procedure as it cannot capture the error signals during back-propagation process [13]. The other one is its high computation time complexity which is linear to the size of input sequence. To overcome its first drawback, more advanced variants have been proposed. The most successful ones are gated recurrent unit(GRU) [20] and long short-term memory(LSTM) [57].

GRU GRU also utilize gating mechanism to control the information flow. Specifically, in the GRU, current state $\mathbf{h}_t$ is a linear interpolation between previous state $\mathbf{h}_{t-1}$ and a candidate state $\tilde{\mathbf{h}}_t$:

$$\mathbf{h}_t = z_t \odot \mathbf{h}_{t-1} + (1 - z_t) \odot \tilde{\mathbf{h}}_t \tag{2.2}$$

where $\odot$ is the element-wise multiplication and z_t is called update gate which is introduced to control how much current state should be updated. It is obtained through the following equation:

$$z_t = \sigma(\mathbf{W}_z \mathbf{x}_t + \mathbf{U}_z \mathbf{h}_{t-1}) \tag{2.3}$$

Where $\mathbf{W}_z$ and $\mathbf{U}_z$ are the parameters and $\sigma(\cdot)$ is the sigmoid function, that is, $\sigma(x) = \frac{1}{1+e^{-x}}$. In addition, the newly introduced candidate state $\tilde{\mathbf{h}}_\mathbf{t}$ is computed by the Equation 2.4:

$$\tilde{h}_t = g(\mathbf{W}\mathbf{x}_t + \mathbf{U}(r_t \odot \mathbf{h}_{t-1})) \tag{2.4}$$

where $g(\cdot)$ is the Tanh function that $g(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ and $\mathbf{W}$ and $\mathbf{U}$ are model parameters. r_t is the reset gate which determines the contribution of previous state to the candidate state and is obtained as follows:

$$r_t = \sigma(\mathbf{W}_r \mathbf{x}_t + \mathbf{U}_r \mathbf{h}_{t-1}) \tag{2.5}$$

LSTM. Similar to GRU, LSTM also utilize gating mechanism to control the information flow. However, comparing to GRU, the structure of LSTM is more complex, which has four gates and an additional cell state. More concretely, the LSTM is defined as follows:

$$
\begin{aligned}
f_t &= \sigma_g(\mathbf{W}_f \mathbf{x}_t + \mathbf{U}_f \mathbf{h}_{t-1} + b_f) \\
i_t &= \sigma_g(\mathbf{W}_i \mathbf{x}_t + \mathbf{U}_i \mathbf{h}_{t-1} + b_i) \\
o_t &= \sigma_g(\mathbf{W}_o \mathbf{x}_t + \mathbf{U}_o \mathbf{h}_{t-1} + b_o) \\
\tilde{\mathbf{c}}_t &= \sigma_c(\mathbf{W}_c \mathbf{x}_t + \mathbf{U}_c \mathbf{h}_{t-1} + b_c) \\
\mathbf{c}_t &= f_t \odot \mathbf{c}_{t-1} + i_t \odot \tilde{\mathbf{c}}_t \\
\mathbf{h}_t &= o_t \odot \sigma_h(\mathbf{c}_t)
\end{aligned}
\tag{2.6}
$$

where f_t, i_t and o_t are the forget, input and output gate, respectively. $\mathbf{W}_*$, $\mathbf{U}_*$ and b_* ($* \in \{f, i, o, c\}$) are trainable parameters. Despite that both LSTM and GRU alleviate the problems of gradient exploding/vanishing, they still suffers from slow computation due to the recurrent procedure. In the next subsections, sequence models without any recurrent procedure are introduced.

2.1.3 Transformer Model

Since its first appearance, the Transformer model has grained great success in many fields [117]. The major building material of Transformer model is the attention mechanism that was introduced to alleviate the bottleneck brought by a fixed-length context vector in machine translation problem [8]. In a nutshell, Transformer consists of a encoder and a decoder. Since only encoder is related to our works, we will mainly focus on it and encourage readers to refer the original paper for details of decoder.

The input of the Transformer encoder are the event embeddings, to which positional encodings will be added. After this, the input will be passed through stacked identical blocks. Each block consists a multi-head attention and a feedforward sublayers that are connected with residual and layer norm operations [53, 7]. The multi-head attention sublayer is the most important structure of encoder. Roughly speaking, the multi-head attention layer allows every event to attend all events and refines the representation of each event as the weighted sum of all events. Comparing to RNNs, there are two major advantages of Transformer encoder. The first one is its effectiveness for long-term dependencies, which is attributed to the direction connection between events in the multi-head attention sublayer. The second is the ease of parallelization of computation. The computation for representations of each event is independent of that of any others. Therefore, the time complexity will not changed by the input size.

2.2 Sequence Modeling Problems

In this section, we will introduce sequence modeling problems which can be roughly grouped into two categories, that is, sequence classification and next event prediction.

2.2.1 Sequence Classification

The goal of this problem is to learn a sequence model that can map unseen sequences to their labels. Depending on the definitions of sequence and label, this problem corresponds to a various of tasks. For example, if the sequences are sentences and the label is sentiment, i.e., positive/negative, then it is sentiment classification task [108]. Another example would be the spam detection task where the sequence is a email and the label is whether the email is spam or not [64].

2.2.2 Next Event Prediction

In this problem, the model aims to predict the event in next step given the a sequence contains all the events seen so far. One typical task is the language modeling where the sequence is a prefix of a word/sentence, we want to predict the next immediately character/word [106]. Comparing to the sequence classification problem, next event prediction often relies more on the model's ability to preserve the historical information.

2.3 Summary

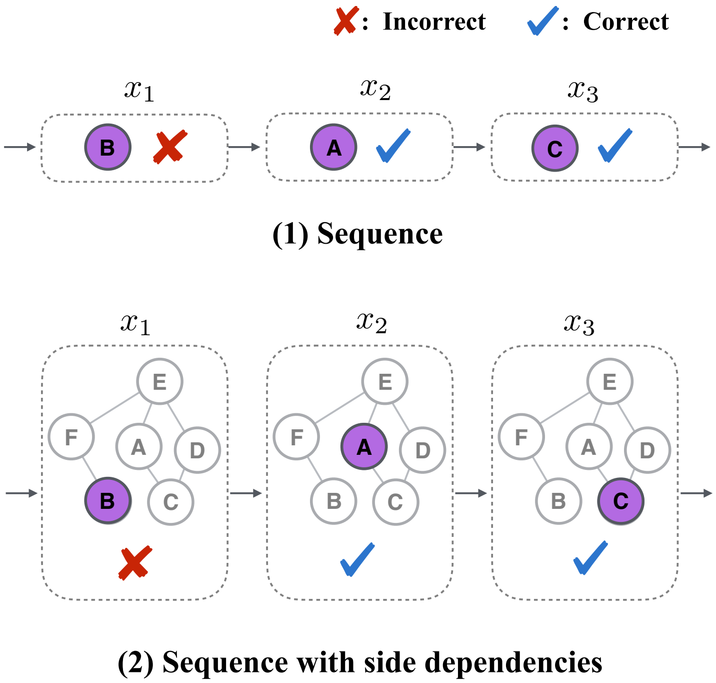
This chapter covers the knowledge background of sequence modeling including popular neural networks and common tasks. It helps to lay the foundation for the next chapters which present our effort to tackle new challenges in sequence modeling.

Chapter 3

Modeling Event Relation

3.1 Introduction

Besides sequential dependencies, the real-world sequences of events often exhibit side relation that may not fully reflected in the sequences. In the next, I will give a few examples. In session-based recommendation [56], items involved in sessions can be related with each other due to the factors such as their brands, the categories they belong to, and prices, etc. In the knowledge tracing scenario [97] where we want to model the knowledge state of a student based on her historic interactions with questions, questions can be related because they are designed to examine similar knowledge or skills behind them. In the document modeling task [133], words can have similar features such as their syntactic functions. The event relations can be naturally captured by a side graph where nodes are events in the sequences and there is an edge between two events if they are related. Figure 3.1 illustrate a toy example about sequences with and without side relations in the knowledge tracing task. In this example, there are total 6 questions ($\{A, B, C, D, E, F\}$) in the database and a student has 3 sequential events ($\{e_1, e_2, e_3\}$) where each event includes the question the student answered and the correctness of the answer. Compared to the sequence without event relation in Figure 3.1 (1), the 6 questions are related by the skills required to solve them and the relation is modeled by a question-question graph in Figure 3.1 (2). Given that the student



(1) Sequence

(2) Sequence with side dependencies

Figure 3.1: A toy example of a sequence with and without side dependencies in the knowledge tracing task. Note that each square indicates an interaction where the purple node denoting the question, and cross and check marks denote the correctness of the student's answer.

had correctly answered the question “A" in e_2, it could be suggested that the student is likely to correctly answer the question “C" since they require similar skills as indicated by the question-question graph. Thus, incorporating the event relation in addition to sequential dependencies has the great potential to advance the modeling of sequences.

While existing models are effective in capturing the sequential dependencies, the majority of them cannot take advantage of the event relation that naturally exist among events of sequences. Thus, dedicated efforts to develop more advanced sequence learning models are needed. However, several obstacles make exploiting event relations for sequence learning a challenging problem. First, sequential dependencies and event relations are inherently different that sequential dependencies rely totally on the sequential data while event relations capture intrinsic properties of events. Thus, it can be very difficult to incorporate them simultaneously. Meanwhile, sequential dependencies and event relations can have varied contributions in sequence modeling tasks. Therefore, how to prevent one from being dominated by the other one during learning process is another obstacle. One straightforward strategy to incorporate the event relations in the sequence learning is to apply graph embedding algorithms on the side graph such as LINE [109] and Node2vect [44] to obtain a low-dimension representation vector for each event, which can be utilized as attributes of events in the existing sequence learning models. However, this solution could be suboptimal since the network embedding space may not align well with the sequence representation space. Thus, we investigate the problem of modeling sequences with event relations. In an attempt to solve the aforementioned obstacles, we propose a novel sequence learning framework that captures both sequential and event relations simultaneously in an end-to-end approach.

3.2 Problem Statement

In this section, we will formally define the problem we aim to solve. Let $\mathcal{E} = \{e^1, e^2, e^3, \cdots, e^n\}$ be a event set with n events in total and a sequence S be denoted as $S = (x_1, x_2, \cdots, x_i \cdots, x_t, x_{t+1})$, where x_i is referred as the i^{th} event. In addition, each event x_i in a sequence S involves one event from $\mathcal{E}$, denoted as g^i. In many real-world applications, events in $\mathcal{E}$ exhibit some intrinsic relations that are captured as a weighted undirected graph $\mathcal{G}$. Following the standard way to represent graphs, $\mathcal{G}$ is denoted by an adjacent matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$, where $\mathbf{A}_{i,j}$ indicates the strength of the relation between events e^i and e^j.

Most of sequence learning tasks follow the supervised learning settings that each sequence S^i in the sequence training set $\mathcal{S}_L = \{(S^i, y^i)\}_1^N$ is associated with a label y^i. The sequence learning aims to utilize information in $\mathcal{S}_L = \{(S^i, y^i)\}_1^K$ and predict the labels for sequences without labels. Depending on the specific tasks, the definitions of labels are different. For example, in natural language process, labels can be the sentiment of a sentence [30]; in computer vision, labels can be the content of images [119]; and in session-based recommendations, labels can be the sentiment of users on items [56]. In this work, we focus on user behavior modeling tasks where the labels are the user's actions on a given event in $\mathcal{E}$ as shown in Figure 3.1. In particular, for each sequence $S^i = [x_1, x_2, \cdots, \cdots, x_t, x_{t+1}]$ in the training set, the event $x_j = (g^j, y^j)$ for $j \in \{1, 2, \cdots, t\}$ where y^j denotes the action on g^j and the label $y^i = y^{t+1}$. In other words, given the actions on events in events from 1 to t, the task we are investigating is to predict the action on the event g^{t+1} in the next event x_{t+1}. This is an essential task in a wide range of real-world applications. For instance, in recommendations, this task is equivalent to predict the actions of users on the next recommended item; and in Education, it is the task of knowledge tracing that aims to predict whether a user gives correct or wrong answer to the next question. Note that given the generalization of the sequence modeling component of the proposed framework in this work, it is straightforward to extend the framework to other tasks and we will leave it as one future work.

With the above notations and definitions, the problem we aim to solve in this work is formally defined as follows:

Given an event set $\mathcal{E}$ with event relations in $\mathbf{A}$ and a labeled sequence set $\mathcal{S}_L = \{(S^i, y^i)\}_1^K$, where each sequence $S^i = [x_1, x_2, \cdots, x_t, x_{t+1}]$ is generated by a specific user u^i and each event $x_t = (g^t, y^t)$ involves one event $g^t \in \mathcal{E}$ and the action label y^i, our goal is to build a framework that is able to predict the labels of unlabeled sequences by capturing both sequential and event relations.

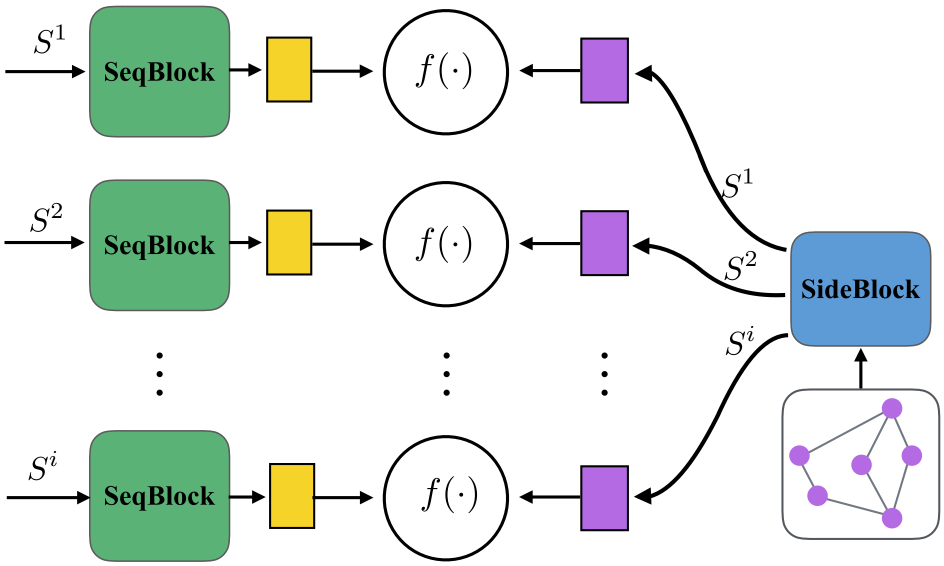


Figure 3.2: The overall structure of the proposed framework. It consists of two major components: *SeqBlock* and *SideBlock*

3.3 The Proposed Framework

In real sequential data, events naturally exhibit intrinsic relations in addition to the sequential dependencies. In this section, we propose a framework that is able to take advantage of event relations for sequence learning. The overall structure of the framework is shown in Figure 3.2. It consists of two major components: *SeqBlock* and *SideBlock*, which capture the sequential dependencies and event relations, respectively. Specifically, the *SeqBlock* takes a sequence as input and output a latent representation vector. In addition, the *SideBlock* will take the event relations graph as the input and output event representation vectors, which will be queried by event sequences. Finally, the framework will predict the user action label based on the sequence representation and the queried event representation. In the next subsections, we will describe each component in details.

3.3.1 Learning Sequential Dependencies

Previously, sequence learning relies on Recurrent Neural Networks (RNN) as fundamental modules. Many variants based on RNN have been proposed in order to solve the notoriously difficult problem: capturing the long-term sequential dependencies. Among them, structures with gated mechanism such as Long Term Short Memory [57] and Gated Recurrent Unit [20] stand out due to their widely received success [23]. However, RNN based models suffer from the inefficiency issue because of its sequential nature, which makes them very difficult to parallel within-sequence computation. To overcome this issue, many efforts have been devoted to developing models without recurrent structures, such as temporal convolutional networks, which can have similar performance with RNN based models while having much faster computation speed [10]. Recently, promising results for a wide range of sequence learning tasks have been obtained by models that heavily relies on attention mechanism including *memory networks* and *Transformer* [117, 105]. Inspired by aforementioned advances, we develop the sequence dependency component, which we refer as *SeqBlock*.

3.3.1.1 The Input Layer

The input of the *Seqblock* is a sequence of events $(x_1, x_2, \cdots, x_t)$. Each event x_j is denoted by three types of information – the corresponding event g^j, the action label y^j and the sequential order information in the sequence. We use a vector $\mathbf{g}^j \in \mathbb{R}^{d/2}$ to indicate the event. To incorporate the action information, we define the event vector for x_j as follows:

$$\mathbf{x}_j = \begin{cases} [\mathbf{g}_j \| \mathbf{0}], & y^j = 1 \\ [\mathbf{0} \| \mathbf{g}_j], & y^j = -1 \end{cases} \tag{3.1}$$

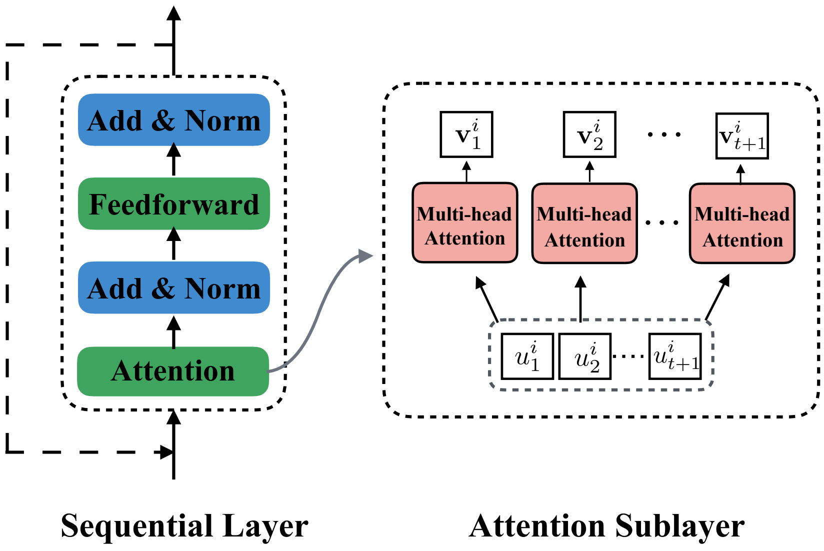
where $\|$ is the concatenation operation, $\mathbf{g}_j = \mathbf{e}^i$ if g^j is the i^{th} event in $\mathcal{E}$ and $\mathbf{0} \in \mathbb{R}^{d/2}$ is a zero vector. $\mathbf{e}^i$ is the embedding of the i-th event in $\mathcal{E}$, which can be pre-trained or randomly initialized. Note that in this work, we consider binary labels and it can be naturally extended for the multiple labels scenario. In addition, we want to embed the order information in the event vector. As suggested by the previous work [40, 117], we add the order information into $\mathbf{x}_j$ as:

$$\mathbf{x}_j(i) = \begin{cases} \mathbf{x}_j(i) + sin(\frac{j}{10000^{i/d}}), & i\%2 = 0 \\ \mathbf{x}_j(i) + cos(\frac{j}{10000^{(i-1)/d}}), & otherwise \end{cases} \tag{3.2}$$

3.3.1.2 The Sequential Layer

Inspired by previous works [117, 105], we utilize attention mechanism to capture the sequential dependency among events. As shown in Figure 3.3, each sequential layer consists of two sublayers – one attention sublayer and one feedforward sublayer and there could be multiple sequential layers stacking together. The input of the first sequential layer is the output of the event layer plus one special vector $\mathbf{x}_{t+1}$, which is randomly initialized and its corresponding final state will be used as the representation of the whole sequence. Next, without the loss of generality, we will illustrate the i^{th} sequential layer and assume the input vectors of the i^{th} sequential layer are $\mathbf{u}^i_j, j \in \{1, 2, \cdots, t+1\}$. Since the output of the event layer is the input of the first sequential layer, $\mathbf{u}^1_j = \mathbf{x}_j$. Next we introduce the attention sublayer and the feedforward sublayer in the i^{th} sequential layer.

Attention Sublayer: The attention sublayer is shown in right subfigure of the Figure 3.3. The input of the attention sublayer are $\mathbf{u}^i_j, j \in \{1, 2, \cdots, t+1\}$. For each input vector $\mathbf{u}^i_j$, the corresponding output vector of the attention layer is $\mathbf{v}^i_j$, which is obtained through



Sequential Layer **Attention Sublayer**

Figure 3.3: The structure of *SeqBlock* is shown in the left subfigure. It consists of two sublayers: one attention sublayer and one feedforward sublayer. The right subfigure shows the structure of the attention sublayer.

multihead mechanism, which is design to capture the event sequential dependencies from different aspects. Next, we will use $\mathbf{v}_m^i$ to illustrate the multihead mechanism.

The overall structure of the multihead mechanism is demonstrated in Figure 3.4. Specifically, we assume that there are n^a attention heads, each of which corresponds to a set of query, key, and value spaces. In the $head_k$, to obtain the attention score of each input vector with respect to $\mathbf{u}_m^i$, the input vectors are projected into the key space and $\mathbf{u}_m^i$ is projected into the query space. Specifically, let α_j^{ik} be the attention score of $\mathbf{u}_j^i$ with respect to $\mathbf{u}_m^i$ in $head_k$. Then α_j^{ik} is calculated as follows:

$$\alpha_j^{ik} = softmax(\frac{\mathbf{u}_m^i\mathbf{W}^{Qk} \cdot \mathbf{u}_j^i\mathbf{W}^{Kk}}{\sqrt{d}\sum_{l=1}^{t+1}\mathbf{u}_m^i\mathbf{W}^{Qk} \cdot \mathbf{u}_l^i\mathbf{W}^{Kk}}) \qquad (3.3)$$

where $\mathbf{W}^{Qk} \in \mathbb{R}^{d\times d}$ and $\mathbf{W}^{Kk} \in \mathbb{R}^{d\times d}$ are the query and key projection matrices in $head_k$, respectively. and $\cdot$ is the dot product operator. With the obtained attention scores, the new presentation of $\mathbf{u}_m^i$ in $head_k$ can be calculated as the weighed sum of all the input vectors in the value space. Let $\mathbf{h}_m^{ik}$ be the new representation of $\mathbf{u}_m^i$ in $head_k$:

$$\mathbf{h}_m^{ik} = \sum_{l=1}^{t+1} \alpha_l^{ik} \mathbf{u}_l^i \mathbf{W}^{Vk} \tag{3.4}$$

where $\mathbf{W}^{Vi} \in \mathbb{R}^{d \times d^v}$ is the project matrix that maps the input vector into the value space. In this way, we are able to obtain the new representation of $\mathbf{u}_m^{ik}$ in each head. Thus, given $\mathbf{h}_m^{ik}, k \in \{1, 2, \cdots, n^a\}$, $\mathbf{v}_m^i$ can be calculated as follows:

$$\mathbf{v}_m^i = [\mathbf{h}_m^{i1} \| \mathbf{h}_m^{i2} \| \cdots \| \mathbf{h}_m^{in^a}] \mathbf{W}^O \tag{3.5}$$

where $\mathbf{W}^O \in \mathbb{R}^{d^v n^a \times d}$ is another projection matrix. The multihead attention mechanism not only allows to learn sequential dependency of events effectively, but also enables the calculations of Equation 3.4 be proceeded parallelly, which could considerably reduce the time cost of the training procedure. As we previously mentioned that the sequential layer can be stacked together to form a very deep structure. Thus, to reduce the training difficulty when the structure goes deep, following previous work [53], we add a shortcut connection for the attention sublayer followed by the layer normalization [7]. Specifically, given the $\mathbf{v}_j^i, j \in \{1, 2, \cdots, t+1\}$, the short connection and layer normalization operation output the vector $\tilde{\mathbf{v}_j^i}$ as follows:

$$\tilde{\mathbf{v}_j^i} = LayerNorm(\mathbf{v}_j^i + \mathbf{u}_j^i) \tag{3.6}$$

where we define the $LayerNorm(\cdot)$ function similar to that in [7].

Feedward sublayer: The second sublayer consists of a fully connected feed-forward network (FFN). Its input is $\tilde{\mathbf{v}_j^i}, j \in \{1, 2, \cdots, t+1\}$. The output of the feedward sublayer for $\tilde{\mathbf{v}_j^i}$ is

$\hat{\mathbf{v}}_j^i$, that is calculated as follows:

$$\hat{\mathbf{v}}_j^i = FFN(\tilde{\mathbf{v}}_j^i) = ReLU(\tilde{\mathbf{v}}_j^i \mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + b_2 \tag{3.7}$$

where $\mathbf{W}_1 \in \mathbb{R}^{d \times d^f}$, $\mathbf{W}_2 \in \mathbb{R}^{d^f \times d}$, $\mathbf{b}_1 \in \mathbb{R}^{d^f}$, and $\mathbf{b}_2 \in \mathbb{R}^d$ are the learnable parameters and $RelU(cdot)$ is the non-linear activation function. Similarly, the short connection and layer normalization operation are also applied to the feedward sublayer, which will produce $\mathbf{u}_j^{i+1}, j \in \{1, 2, \cdots, t+1\}$ corresponding to $\hat{\mathbf{v}}_j^i$. Assume that there are n^{sl} sequential layers stacking together, the $t + 1^{th}$ output vector of the last layer $\mathbf{u}_{t+1}^{n^{sl}+1}$ will be used as the representation of the whole sequence.

In summary, the *SeqBlock* takes the event sequence S as the input and outputs a representation of the sequence $\mathbf{u}_{t+1}^{n^{sl}+1}$. Although the *SeqBlock* effectively captures the sequential dependency of the events, it completely ignores the side dependencies that are not reflected in the sequences. Next, we will introduce the model component to incorporate the event relations.

3.3.2 Capturing Event Relations

In this subsection, we introduce the model component *SideBlock* of the framework and it aims to capture the event relations.

One straightforward solution to incorporate the relations of events is to apply graph embedding algorithms to the event-event graph $\mathcal{G}$ to obtain an embedding vector that contains the relation information for each event. For instance, the state-of-the-art network embedding algorithms such as LINE [109] and Node2Vec [44], can map the nodes in a graph to a subspace that preserves maximum neighborhood information of nodes. Thus, the event embedding

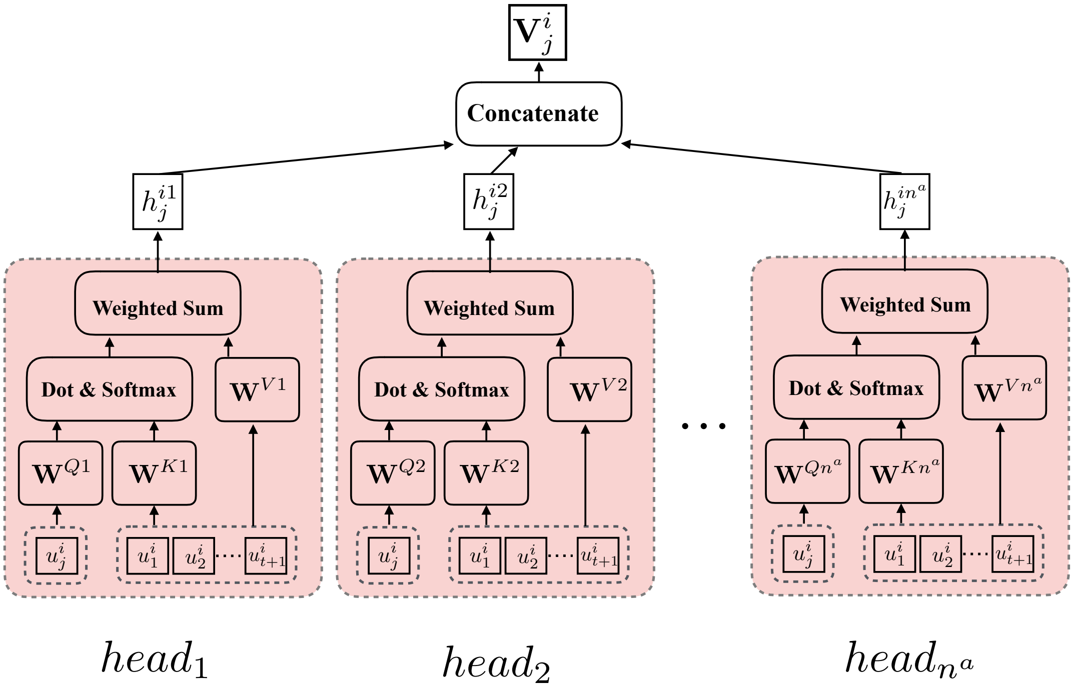


Figure 3.4: The illustrative example of multihead attention mechanism.

obtained by such algorithm naturally carries their relation information. Then we can use event graph embeddings to be the event representation vectors, which can be inputted in *SeqBlock*. In fact, this solution is similar to one widely used approach in natural language processing tasks where pre-trained word embeddings that capture the word relations through large corpuses are utilized to improve the performance [18, 24, 68]. However, this solution may be not sufficient to fully take advantage of event relations because: 1) as the sequential learning block can go very deep and can completely focus on modeling the sequential dependencies, it is likely that sequential dependency will dominate event relations; and 2) the embedding of events is obtained by graph embedding algorithms that run in a separate procedure, thus they may not be optimal for the specific task. Thus, we propose *SideBlock* to effectively capture the event relations. Specifically, it consists of two layers: one linear projection layer and one aggregation layer. Next we will describe each layer.

3.3.2.1 Linear Project Layer

The input of the linear projection layer is the event embedding vectors $\mathbf{e}^j, j \in \{1, 2, \cdots, n\}$. There are many ways to compute $\mathbf{e}^j$. For example, $\mathbf{e}^j$ can be sampled from a normal distribution $\mathcal{N}(0, \mathbf{I})$ and it only serves as the indicator of the i^th event; while it can be also obtained from the output of the graph embedding algorithms. Given $\mathbf{e}^j$, the linear projection function in this layer is defined as:

$$\hat{\mathbf{e}}^j = \mathbf{e}^j \mathbf{W}^P + \mathbf{b}^P \tag{3.8}$$

where $\mathbf{W}^P \in \mathbb{R}^{d^e \times d}$ and $\mathbf{b}^P$ are the projection matrix and the bias term, respectively. $\hat{\mathbf{e}}^j \in \mathbb{R}^d$ is the projected vector of the j^{th} event. As we stated previously, the embedding space may not align well with the sequence representation space. Therefore, the linear project layer will enable the model to learn to project the embedding vectors into a space where the aggregation layer is designed to preserve the event relations as shown next.

3.3.2.2 Aggregation Layer

Intuitively, if an event is related to its neighbors in the graph $\mathcal{G}$, it is desirable that its representation is also close to that of its neighbors. For instance, in the knowledge tracing scenario, questions requiring similar skills should be located in a similar spot in the representation space. To impose this intuition, inspired by previous work [48, 71], we design 3 sublayers in the aggregation layer – feedforward sublayer, neighbor sublayer and integration sublayer. Note that, the aggregation layer can also be stacked to a deep structure. Without the loss of generality, next we detail the i^{th} aggregation layer. Let the input of the i^{th} aggregation layer is $\mathbf{c}^{ij}, j \in \{1, 2, \cdots, n\}$. Thus, $\mathbf{c}^{1j} = \hat{\mathbf{e}}^j$.

The input of the feedforward layer are $\mathbf{c}^{ij}, j \in \{1, 2, \cdots, n\}$. For each $\mathbf{c}^{ij}$, the feedforward will output a vector $\hat{\mathbf{c}}^{ij}$, which can be calculated as:

$$\hat{\mathbf{c}}^{ij} = ReLU(\mathbf{c}^{ij}\mathbf{W}^{a1} + \mathbf{b}^{a1}) \tag{3.9}$$

where $\mathbf{W}^{a1} \in \mathbb{R}^{d \times d}$ and $\mathbf{b}^{a1} \in \mathbb{R}^{d}$are learnable parameters. This feedforward layer aims to learn better representation of events in the latent space. The second sublayer is to aggregate the neighbors' information. The input is $\hat{\mathbf{c}}^{ij}, j \in \{1, 2, \cdots, n\}$ and for each $\hat{\mathbf{c}}^{ij}$, this sublayer will also output a vector $\tilde{\mathbf{c}}^{ij}$ which is defined as:

$$\tilde{\mathbf{c}}^{ij} = \sum_{l \in \mathcal{N}^j} \hat{\mathbf{A}}(j, l)\hat{\mathbf{c}}^{il} \tag{3.10}$$

where $\mathcal{N}^j$ is the neighborhood set of j^{th} events and $\hat{\mathbf{A}}$ is the normalized adjacent matrix for $\mathcal{G}$, which is defined as:

$$\hat{\mathbf{A}} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2} \tag{3.11}$$

where $\mathbf{D}$ is the diagonal matrix and $\mathbf{D}(i, i) = \sum_j \mathbf{A}(i, j)$. With the latent vector $\tilde{\mathbf{c}}^{ij}$ that captures neighborhood information of the j^{th} event, the third sublayer is to integrate $\tilde{\mathbf{c}}^{ij}$ and $\hat{\mathbf{c}}^{ij}$ to obtain the final representation of the event, which is defined as follows:

$$\mathbf{c}^{(i+1)j} = ReLU([\tilde{\mathbf{c}}^{ij} \| \hat{\mathbf{c}}^{ij}]\mathbf{W}^{a2} + \mathbf{b}^{a2}) \tag{3.12}$$

where $\mathbf{W}^{a2} \in \mathbb{R}^{2d \times d}$ and $\mathbf{b}^{a2} \in \mathbb{R}^{d}$ are the model parameters. Let n^{ag} to be the total number of aggregation layers. The output of the *SideBlock* will be $\mathbf{c}^{n^{ag}j}, j \in \{1, 2, \cdots, n\}$.

3.3.3 The Objective Function and Training

With the previously described model components that are able to capture both sequential dependencies and event relations, the proposed framework SEE is formulated as follows:

$$\mathbf{u}_{t+1}^{n^{sl}} = \mathit{SeqBlock}(S) \tag{3.13}$$

$$\{\mathbf{c}^{n^{ag}1}, \mathbf{c}^{n^{ag}2}, \cdots, \mathbf{c}^{n^{ag}n}\} = \mathit{SideBlock}(\{\mathbf{e}^1, \mathbf{e}^2, \cdots, \mathbf{e}^n\}, A) \tag{3.14}$$

$$\mathbf{z} = \mathit{query}(S, \{\mathbf{c}^{n^{ag}1}, \mathbf{c}^{n^{ag}2}, \cdots, \mathbf{c}^{n^{ag}n}\}) \tag{3.15}$$

$$\hat{y} = f(\mathbf{u}_{t+1}^{n^{sl}}, \mathbf{z}) \tag{3.16}$$

where $S = (x_1, x_2, \cdots, x_t)$ is the input sequence and the *SeqBlock* will produce $\mathbf{u}_{t+1}^{nsl}$ that embeds the sequential information of the sequence. $\{\mathbf{e}^1, \mathbf{e}^2, \cdots, \mathbf{e}^n\}$ are the initial event embedding vectors and the *SideBlock* will produce a new set of representation vectors for events in the latent space that align well with the sequence representation space. In addition, S will query one event vector from $\{\mathbf{c}^{n^{ag}1}, \mathbf{c}^{n^{ag}2}, \cdots, \mathbf{c}^{n^{ag}n}$ according to g_{t+1}. Specifically, $\mathbf{z} = \mathbf{c}^{n^{ag}j}$ if g_{t+1} is the j^{th} event in $\mathcal{E}$. Finally, $f(\cdot)$ is the prediction layer and $\hat{y}$ is the predicted action on the next event g_{t+1}. There are many choices for $f(\cdot)$ and in this work, we define it as follows:

$$f(\mathbf{u}_{t+1}^{n^{sl}}, \mathbf{z}) = \sigma(\mathbf{u}_{t+1}^{n^{sl}} \cdot \mathbf{z}) \tag{3.17}$$

where $\sigma(\cdot)$ is the Sigmoid function. Then we define the following loss function to train the framework:

$$\mathcal{L} = \sum_{j=1}^{K} -y^j \log(\hat{y}^j) - (1 - y^j) log(1 - \hat{y}^j) \tag{3.18}$$

where K in the number of sequences in the training set, $\hat{y}^j$ and y^j are the predicted and true labels for the j^{th} sequence, respectively.

In order to effectively train the proposed framework SEE, we exploit the widely used Truncated Back Propagation Trough Time (TBPTT) algorithm. Specifically, we firstly define M to be the number of timesteps that are used for both forward and back-pass procedure. Then, for each sequence, we take M consecutive samples $(x_k, y_k)_{k=j}^{k+M}$. Then we apply the forward-pass of the framework to these samples and obtain a error signal. Finally, according to the error, the back-pass will update the parameters of the proposed framework.

3.4 Experiment

In this section, we evaluate the proposed framework SEE with real sequential data. In the following subsections, we first describe the two tasks and the corresponding datasets to assess the performance of sequence learning. Then we present and analyze the experimental results. Finally, we conduct model component analysis to gain a deeper understanding of the proposed framework SEE.

3.4.1 Experimental Settings

In this work, we choose two tasks from two different domains to evaluate the performance of the proposed framework, i.e., knowledge tracing and session-based recommendations.

3.4.1.1 Knowledge Tracing

In the education field, knowledge tracing is to monitor student knowledge states and skill acquisition levels, which is highly important for personalized education [97, 19]. One of the key tasks of knowledge tracing is to predict a student's future interactions with questions in the question database based on her past interactions, where an interaction denotes that the student answers one question. Thus, it is easily to formulate this task as the user behavior sequence modeling problem stated in Section 2. Specifically, we regard past interactions as the event sequence, where each question is an event and the correctness of the student's answer is the action. Then the task is to predict whether the student gives correct or wrong answer to the next question. To conduct experiment on this task, we collect a GMAT dataset from a GMAT preparation mobile application [1], which is one of the most popular apps in this kind in China. This dataset describes students' interactions with GMAT questions that are designed to improve students' problem-solve skills. All the student identification information is removed. Specifically, it contains 90831 students and their 16002324 records with 8684 questions. Moreover, each interaction record consists of one student ID, one question ID, correctness of the student answer, and the time stamp of this interaction. Thus, for each student, a sequence of interactions can be formed according to the time stamp of each interaction. We filter those students who have only few interactions since we are not focusing on the "cold start" problem. In addition, we also remove the questions that are not

[1] http://www.kmf.com/

in the interactions of the remaining students. After such filtering procedure, there are 20,251 sequences and 7,530 questions. We then construct a weighted question event relations graph according to the knowledge and skills required by questions. We further randomly split the sequences into training, validation and testing sets such that each interaction sequence will be put in one and only one of the three sets. As results, the training, validation, and test sets have 14,176, 3,038, 3,037 sequences, respectively.

3.4.1.2 Session-based Recommendation

Session-based recommendation is getting increasingly popular in recommender systems community. It aims to recommend items by modeling user's preference from their past interactions. To achieve this goal, most of previous works[56, 127, 143], have chosen RNN based models to build the recommender systems, which tend to ignore relations of items. In this subsection, we easily apply the proposed framework SEE to solve the recommendation task. The recommendation task can also be formulated as our previously defined problem with items being the events and user rating being the action. To evaluate the proposed framework on this task, we collect the MovieLens dataset, which is publicly available [2] and has been used for many recommendation related works [51, 141, 58, 78]. This dataset describes user preferences for movies. Specifically, it contains 200 million ratings for movies from 138,493 users. Each rating is a 10-scale score and in this work, we convert the rating to a binary score indicating whether the user likes a movie or not. Specifically, $score = 1$ if $rating > 4$, otherwise $score = 0$. Each rating is also associated with a time stamp, according to which all of ratings given by one user can be arranged as one sequence. Moreover, it also provides the additional information of the movies including their title, genre, release year,

[2] https://grouplens.org/datasets/movielens/

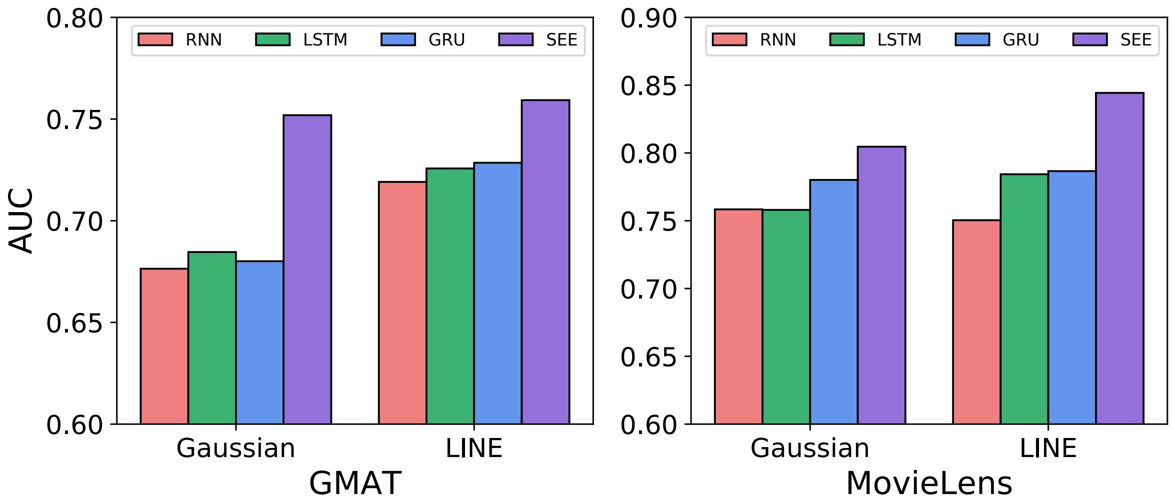


Figure 3.5: Performance comparison for baselines and SEE. Results on knowledge tracing and recommendation tasks are shown in the left and right subfigures, respectively. The y-axis is the AUC score. Higher score means better prediction. For each task, models are grouped by the input vectors – Gaussian indicates that models use vectors sampled from Gaussian distribution to represent events while LINE suggests that the events are represented by vectors obtained through LINE.

etc. We apply similar filtering procedure to this dataset that removes users who give few ratings and movies that receive few ratings. In the end, there are 20,000 users and 8,270 movies left. We also construct a item relation graph for movies according to their shared attributes. All the sequences are split into training, validation, and testing sets. The resulted training, validation and test sets contain 14,000, 3,000, and 3,000 sequences, respectively.

Evaluation Metric: As we want to assess the performance on predicting the action on the next event, we use Area under the curve (AUC) to measure the action prediction performance since it is widely used and especially suitable for the unbalanced data. The higher value of AUC indicates better performance.

3.4.2 Representative Baselines

Recent years have witnessed great success of RNN-based models for sequence learning tasks. Though they are effective in capturing the sequential dependencies, they tend to ignore event relations. Thus, to demonstrate the importance of event relations and verify the effectiveness of SEE, we compare it with the-state-of-art RNN-based models, which can be divided into two groups. The first group only uses sequential information while the second group of models utilize both sequential and event relations information by first performing graph embedding and then taking the embedding of events as input to the sequence learning. The following are details of baselines:

- RNN + Gaussian. This is the recurrent neural network with vanilla cells. At step j, it will output a state $\mathbf{h}_j = g(\mathbf{W}\mathbf{x}_j + \mathbf{U}\mathbf{h}_{j-1} + \mathbf{b})$, where $\mathbf{x}_j$ is the input vector, $g(\cdot)$ is the activation function, $\mathbf{h}_j$ is the output state, and $\mathbf{W}, \mathbf{U}, \mathbf{b}$ are the model parameters. The last state $\mathbf{h}_t$ is regarded as the final representation of the whole sequence and thus is used to predict user action for each event by the following prediction function:

  $$\mathbf{s} = \sigma(\mathbf{h}_t\mathbf{W}^S + \mathbf{b}^S) \tag{3.19}$$

  , where $\mathbf{W}^S \in \mathbb{R}^{d\times n}$ and $\mathbf{b}^S \in \mathbb{R}^n$ are the parameters of the prediction layer. Thus, if the event in x_{t+1} is the i^{th} event, the predicted label $\hat{y}^j = s(i)$. The initial vectors for events are sampled from Gaussian distribution.

- RNN + LINE. It is based on RNN. However, to incorporate the event relations, the initial vectors are obtained by running the graph embedding algorithm LINE on the event-event graph.

- LSTM + Gaussian [57]. LSTM is a variant of Vanilla RNN by introducing gating mechanism that is essential for capturing the long-term dependencies. The initial vectors for events are sampled from Gaussian distribution.

- LSTM + LINE. It uses LSTM to capture sequential information while performing the graph embedding algorithm LINE as initial vectors to capture event relations.

- GRU+ Gaussian [20]. GRU is another popular recurrent network cell that adopts the gating mechanism. We initialize its input from Gaussian distribution.

- GRU+ LINE. It uses LINE to learn embeddings from the event-event graph to initialize the input of GRU. It incorporates sequential and side information in a separate way.

For a fair comparison, we use Equation 3.1 to incorporate the action information into the event vectors for baseline models. Moreover, as all the baseline models process event vectors with the sequential order, it is no need to add the positional embedding into the event vectors.
Implementation: The Gaussian vectors for events are sampled from Gaussian distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$. The embedding vectors are obtained by running the graph embedding algorithm LINE using the code [3] provided by the authors. In particular, we choose the second order proximity version of this algorithm as it consistently has better performance than the first order [109]. Unless specified otherwise, all the models were implemented in Pytorch [4] and mini-batch stochastic Adam optimizer was used during the training procedure [69]. To prevent models from overfitting problem, dropout was used for all the methods [104]. In addition, we applied the early stopping strategy that the training procedure was stopped when there was no improvement in the performance on the validation set for next 10 epochs.

[3] https://github.com/tangjianpku/LINE

[4] https://pytorch.org/

For each model, the hyperparameters including learning rate and dropout rate were chosen according to the performance on the validation set.

3.4.3 Experimental Results

In this subsection, we present and compare the experimental results of baseline methods and SEE on the aforementioned datasets.

The results for knowledge tracing and recommendation tasks are shown in the left and right subsigures of Figure 3.5, respectively. From the figure, we make the following observations:

- In most of the cases, RNN has the worst performance among baseline methods. This is because that LSTM and GRU have gating mechanism which enable them to capture the long-term dependencies. The results are consistent with the findings of previous works [23];

- For most methods, when representing events with embedding vectors obtained from graph embedding algorithm, the performance increases consistently. This clearly shows the contribution of event relations information to sequence modeling.

- It is very interesting to see that when we compare the improvement brought by the graph embedding vectors, the performance gain of RNN-based baseline models in GMAT dataset is more significant than that in the MovieLens dataset, while the improvement for SEE is less significant in the GMAT dataset than in the Movielens dataset. The reason is that the mechanism by which the graph embedding vectors improve the performance is different for baseline RNN-based models and SEE. For baseline models, the graph embeddings contain the sevent relations. Thus if the embedding space aligns better with the sequential representation space as in the case of GMAT dataset, it is easy for RNN-based models to learn the event relations in the latent space. On the contrary, when the embedding space of events does not align well with the sequence representation space as indicated by the MovieLens case, the improvement brought by the dependency information is limited. On the other hand, SEE is a unified framework to incorporate both information where a latent embedding space which aligns well with the sequence representation space. Thus, better initialization does not necessarily lead to better performance of SEE as the optimization procedure could also play an important role;

- In all the cases, SEE demonstrates performance gain over the baseline methods by a large margin. We attribute such superior performance to its ability to effectively capture both sequential dependencies and event relations. More details analysis of the contributions of major framework components will be discussed in the following subsection.

To sum up, the results for both knowledge tracing and recommendation tasks have clearly demonstrated the advantage of exploiting event relations and verified the effectiveness of the proposed framework SEE. Next, we will analyze the important components of SEE to gain better understanding of its performance.

3.4.4 Component Analysis

The proposed framework SEE has shown significant performance gain over the baseline models. To understand its effectiveness, in this subsection, we analyze two important components of SEE that are the *SeqBlock* and *SideBlock*, which capture the sequential dependencies and event relations, respectively. To do so, we define the following variants of SEE:

- SEE-*SeqBlock*-LINE: this is the variant that only has the *SeqBlock*. Thus it only captures the sequential dependencies. In addition, it uses vectors sampled from Gaussian distribution as the event representation;

- SEE-*SeqBlock*: this variant is similar to SEE-*SeqBlock*-LINE, but uses graph embedding vectors as the event representation;

- SEE-*SideBlock*: this variant only has *SideBlock*. Due to the lack of *SeqBlock*, we samples a random vector to represent the sequential information for avoiding further changes on the framework. Thus, this variant only captures event relations.

The performance of the model variants on both datasets are shown in Figure 3.6. The following can be observed:

- SEE-*SeqBlock* outperforms SEE-*SeqBlock*-LINE, which is consistent with previous findings and suggests the contribution of event relations again;

- SEE-*SideBlock*-LINE has much better performance than the variant with only *SideBlock* in both datasets. This is expected because the sequential information is no doubt important in sequential learning tasks.

In summary, event relations are important for sequence modeling and can boost the performance significantly if properly captured. In addition, the proposed framework has gained the best performance because of its ability to learn both sequential and event relations effectively.

3.5 Discussion

Event relations contain important relation information of events, which have great potentials to boost sequence modeling performance, but few current sequence learning models are

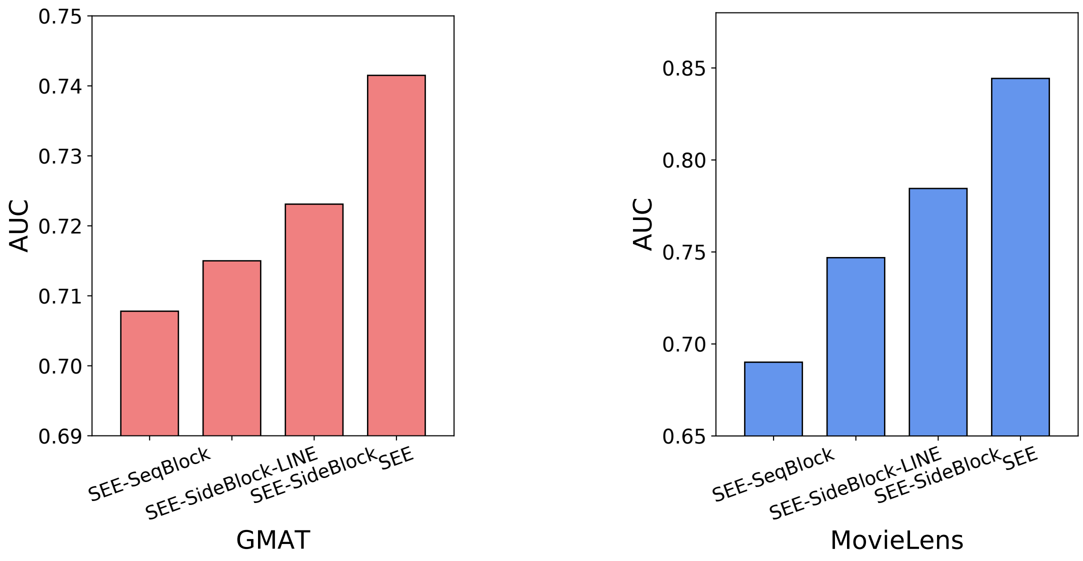


Figure 3.6: Component analysis results

able to capture them. In addition, learning sequential dependencies and event relations simultaneously is challenging because these two types of information are inherently different and it is difficult to prevent one being dominated by the other. In this work, we have exploited event relations for user behavior sequence modeling problem and demonstrated the importance of taking advantage of them. Further, we propose a sequence modeling framework SEE that is able to capture both sequential dependencies and event relations effectively. Evaluating our framework on real data sets that come from totally different domains, we find that the proposed framework outperforms the traditional state-of-the-art sequence learning models significantly.

There are several meaningful future directions to explore. Firstly, our work has focused on user behavior sequence modeling. Thus, one future work could attempt to exploit event relations for other sequential learning tasks such as language modeling. In addition, we believe that event relations information will be even more valuable when data sparsity problem presents, e.g., very few training sequences are available. Therefore, it would be meaningful to utilize it to address cold-start and data sparsity problems. Moreover, our framework and most current sequence learning models assume the events of a sequence is uniformly distributed, which is not true in many cases. For example, in the GMAT data, the time intervals between two consecutive interactions can be very different, which could play an important role in understanding students' knowledge states. Thus, how to incorporate such information in our proposed framework is another meaningful future direction. Finally, in this chapter, we assume the event relations given does not have any noise. However, this assumption may not hold true. In such cases, the event representation obtained by the proposed framework is no longer accurate. Thus, in the future, it is also important to extend our proposed framework to deal with the noise in event relations.

Chapter 4

Modeling Sequence Relation

4.1 Introduction

The majority of existing sequence models such as RNNs have been designed for traditional sequences, which are assumed to be identically, independently distributed (i.i.d.). However, many real-world applications generate linked sequences (sequences are related). For example, web documents, sequences of words, are connected via hyperlinks; genes, sequences of DNA or RNA, typically interact with each other. Figure 4.1 illustrates one toy example of linked sequences where there are four sequences – S^1, S^2, S^3 and S^4. These four sequences are linked via three links – S^2 is connected with S^1 and S^3 and S^3 is linked with S^2 and S^4. On the one hand, these linked sequences are inherently related. For example, linked web documents are likely to be similar [41] and interacted genes tend to share similar functionalities [11]. Hence, linked sequences are not i.i.d., which presents immense challenges to traditional RNNs. On the other hand, linked sequences offer additional sequence relational information in addition to the sequential information. It is evident that sequence relation can be exploited to boost various analytical tasks such as social recommendations [112] , sentiment analysis [121, 59] and feature selection [113]. Thus, the availability of sequence relation information in linked sequences has the great potential to enable us to develop advanced Recurrent Neural Networks.

Now we have established that – (1) traditional RNNs are insufficient and dedicated efforts

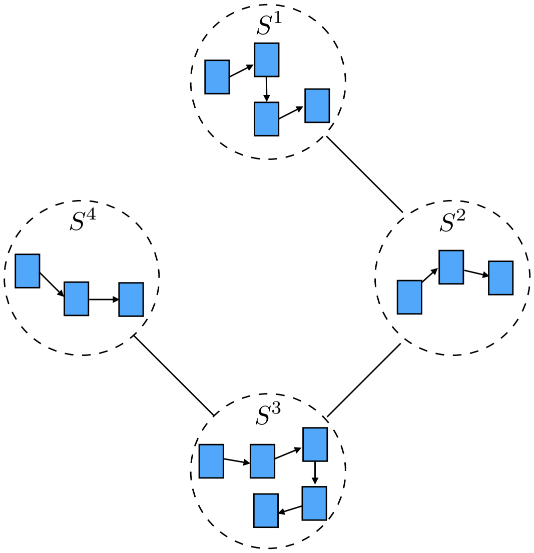


Figure 4.1: An Illustration of Linked Sequences. S^1, S^2, S^3 and S^4 denote four sequences and they are connected via four links.

are needed for linked sequences; and (2) the availability of sequence relations information in linked sequences offer unprecedented opportunities to advance traditional RNNs. In this chapter, we study the problem of modeling sequence relation via RNNs. In particular, we aim to address the following challenges – (1) how to capture sequence relation mathematically and (2) how to combine sequential and sequence relation information via Recurrent Neural Networks. To address these two challenges, we propose a novel Linked Recurrent Neural Network (LinkedRNN) for linked sequences.

4.2 Problem Statement

Let $\mathcal{S} = \{S^1, S^2, \cdots, S^N\}$ be the set of N sequences. For linked sequences, two types of information are available. One is the sequential information for each sequence. We denote the sequential information of S^i as $= (x^i_1, x^i_2, \cdots, x^i_{N^i})$ where N^i is the length of S^i. The other is the sequence relation. We use an adjacent matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ to denote the sequence relation of linked sequences where $\mathbf{A}(i, j) = 1$ if there is a link between the sequence S^i and S^j and $\mathbf{A}(i, j) = 0$, otherwise. In this work, we following the transductive learning setting. In detail, we assume that a part of the sequences from S^1 to S^K are labeled where $K < N$. We denote the labeled sequences as $\mathcal{S}_L = \{S^1, S^2, \ldots, S^K\}$. For a sequence $S^j \in \mathcal{S}_L$, we use y^j to denote its label where y^j is a continuous number for the regression problem and y^j is one symbol for the classification problem. Note that in this work, we focus on the unweighted and undirected links among sequences. However, it is straightforward to extend the proposed framework for weighted and directed links. We would like to leave it as one future work. Although the proposed framework is designed for transductive learning, we also can use it for inductive learning, which will be discussed when we introduce the proposed framework in the following section.

With the above notations and definitions, we formally define the problem we target in this work as follows:

Given a set of sequences $\mathcal{S}$ with sequential information $S^i = (x_1^i, x_2^i, \cdots, x_{N^i}^i)$ and sequence relation $\mathbf{A}$, and a subset of labeled sequences $\{\mathcal{S}_L, (y^j)_{j=1}^K\}$, we aim to build a RNN model by leveraging $\mathcal{S}$, $\mathbf{A}$ and $\{\mathcal{S}_L, (y^j)_{j=1}^K\}$, which can learn representations for sequences to predict the labels of the unlabeled sequences in $\mathcal{S}$.

4.3 The Proposed Framework

In addition to sequential information, sequence relation is available for linked sequences as shown in Figure 4.1. As aforementioned, the major challenges to model linked sequences are how to capture sequence relation and how to combine sequential and relation information coherently. To tackle these two challenges, we propose a novel Recurrent Neural Networks

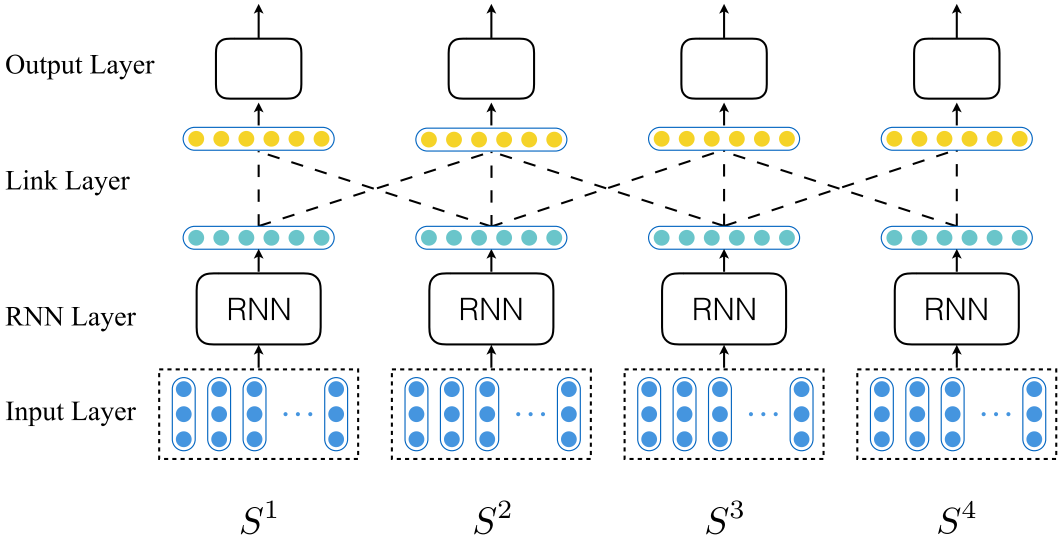


Figure 4.2: An illustrate of the proposed framework LinkedRNN on the toy example as shown in Figure 4.1. It consists of two major layers where RNN layer is to capture sequential information and the link layer is to capture sequence relation.

LinkedRNN. An illustrate of the proposed framework on the toy example of Figure 4.1 is demonstrated in Figure 4.2. It mainly consists of two layers. The RNN layer is to capture the sequential information. The output of the RNN layer is the input of the link layer where sequence relation is captured. Next, we first detail each layer and then present the overall framework of LinkedRNN.

4.3.1 Capturing Sequential Information

Given a sequence $S^i = x^i_1, x^i_2, \cdots x^i_{N^i}$, the RNN layer aims to learn a representation vector that can capture its complex sequential patterns via Recurrent Neural Networks.

In this work, due to its simplicity and effectiveness, we choose GRU as our RNN unit. The details of GRU are described in Section 2. The output of the RNN layer will be the input of the link layer. For a sequence S^i, the RNN layer will learn a sequence of latent representations $(\mathbf{h}^i_1, \mathbf{h}^i_2, \dots, \mathbf{h}^i_{N^i})$. There are various ways to obtain the final output $\hat{\mathbf{h}}_i$ of S^i

from $(\mathbf{h}_1^i, \mathbf{h}_2^i, \ldots, \mathbf{h}_{N^i}^i)$. In this work, we investigate two popular ways:

- As the last latent representation $\mathbf{h}_{N^i}^i$ is able to capture information from previous states, we can just use it as the representation of the whole sequence. We denote this way of aggregation as $aggregation_{11}$. Specifically, we let $\hat{\mathbf{h}}_i = \mathbf{h}_{N^i}^i$.

- The attention mechanism can help the model automatically focus on relevant parts of the sequence to better capture the long-range structure and it has shown effectiveness in many tasks [9, 83, 22]. Thus, we define our second way of aggregation based on the attention mechanism as follows:

$$\hat{\mathbf{h}}_i = \sum_{j=1}^{N^i} a_j \mathbf{h}_j^i \tag{4.1}$$

  where a_j is the attention score, which can be obtained as

$$a_j = \frac{e^{a(\mathbf{h}_j^i)}}{\sum_m e^{a(\mathbf{h}_m^i)}} \tag{4.2}$$

  where $a(\mathbf{h}_j^i)$ is a feedforward layer:

$$a(\mathbf{h}_j^i) = \mathbf{v}_a^T tanh(\mathbf{W}_a \mathbf{h}_j^i) \tag{4.3}$$

  Note that different attention mechanisms can be used, we will leave it as one future work. We denote the aggregation way described above as $aggregation_{12}$.

For the general purpose, we will use RNN to denote GRU in the rest of the chapter.

4.3.2 Capturing Sequence Relation

The RNN layer is able to capture the sequential information. However, in linked sequences, sequences are naturally related. The Homophily theory suggests that linked entities tend to have similar attributes [87], which have been validated in many real-world networks such as social networks [74], web networks [79], and biological networks [11]. As indicated by Homophily, a node is likely to share similar attributes and properties with nodes with connections. In other words, a node is similar to its neighbors. With this intuition, we propose the link layer to capture sequence relation in linked sequences.

As shown in Figure 4.2, to capture sequence relation, for a node, the link layer not only includes information from its sequential information but also aggregates information from its neighbors. The link layer can contain multiple hidden layers. In other words, for one node, we can aggregate information from itself and its neighbors multiple times. Let $\mathbf{v}_i^k$ be the hidden representations of the sequence S^i after k aggregations. Note that when $k = 0$, $\mathbf{v}_i^0$ is the input of the link layer, i.e., $\mathbf{v}_i^0 = \hat{\mathbf{h}}_i$. Then $\mathbf{v}_i^{k+1}$ can be updated as:

$$\mathbf{v}_i^{k+1} = act(\frac{1}{|\mathcal{N}(i)| + 1}(\mathbf{v}_i^k + \sum_{S^j \in \mathcal{N}(i)} \mathbf{v}_j^k)) \tag{4.4}$$

where $act()$ is an element-wise activation function, $\mathcal{N}(i)$ is the set of neighbors who are linked with S^i, i.e., $\mathcal{N}(i) = \{S^j | \mathbf{A}(i,j) = 1\}$, and $|\mathcal{N}(i)|$ is the number of neighbors of S^i. We define $\mathbf{V}^k = [\mathbf{v}_1^k, \mathbf{v}_2^k, \ldots, \mathbf{v}_N^k]$ as the matrix form of representations of all sequences at the k-th layer. We modify the original adjacency matrix $\mathbf{A}$ by allowing $\mathbf{A}(i,i) = 1$. The aggregation in the Equation 4.4 can be written in the matrix form as:

$$\mathbf{V}^{k+1} = act(\mathbf{A}\mathbf{D}^{-1}\mathbf{V}^k) \tag{4.5}$$

where $\mathbf{V}^{k+1}$ is the embedding matrix after $k+1$ step aggregation, and $\mathbf{D}$ is the diagonal matrix where $\mathbf{D}(i,i)$ is defined as:

$$\mathbf{D}(i,i) = \sum_{j=1}^{N} \mathbf{A}(i,j) \tag{4.6}$$

4.3.3 Linked Recurrent Neural Networks

With the model components to capture sequential and relation information, the procedure of the proposed framework LinkedRNN is presented below:

$$\begin{aligned}
&(\mathbf{h}_1^i, \mathbf{h}_2^i, \ldots, \mathbf{h}_{N^i}^i) = RNN(S^i) \\
&\hat{\mathbf{h}}_i = aggregation1(\mathbf{h}_1^i, \mathbf{h}_2^i, \ldots, \mathbf{h}_{N^i}^i) \\
&\mathbf{v}_i^0 = \hat{\mathbf{h}}_i \\
&\mathbf{v}_i^{k+1} = act(\frac{1}{|\mathcal{N}(i)|+1}(\mathbf{v}_i^k + \sum_{S^j \in \mathcal{N}(i)} \mathbf{v}_j^k)) \\
&\mathbf{z}_i = aggregation2(\mathbf{v}_i^0, \mathbf{v}_i^1, \ldots, \mathbf{v}_i^M)
\end{aligned} \tag{4.7}$$

where the input of the RNN layer is the sequential information and the RNN layer will produce the sequence of latent representations $(\mathbf{h}_1^i, \mathbf{h}_2^i, \ldots, \mathbf{h}_{N^i}^i)$. The sequence of latent representations will be aggregated to obtain the output of the RNN layer, which serves as the input of the Link layer. After M layers, link layer produces a sequence of latent representations $(\mathbf{v}_i^0, \mathbf{v}_i^1, \ldots, \mathbf{v}_i^M)$, which will be aggregated to the final representation.

The final representation z_i for the sequence S^i is to aggregate the sequence $(\mathbf{v}_i^0, \mathbf{v}_i^1, \ldots, \mathbf{v}_i^M)$ from the link layer. In this work, we investigate several ways to obtain the final representation $\mathbf{z}_i$ as:

- As $\mathbf{v}_i^M$ is the output of the last layer, we can define the final representation as: $\mathbf{z}_i = \mathbf{v}_i^M$, and we denote this way as $aggregation_{21}$.

- Although the new representation $\mathbf{v}^M$ incorporates all the neighbor information, the signal in the representation of itself may be overwhelmed during the aggregation process. This is especially likely to happen when there are a large number of neighbors. Thus, to make the new representation to focus more on itself, we propose to use a feed forward neural network to perform the combination. We concatenate representations from the last two layers as the input of the feed forward network. We refer this aggregation method as $aggregation_{22}$.

- Each representation $\mathbf{v}_i^j$ could contain its unique information, which cannot be carried in the later part. Thus, similarly, we use a feed forward neural network to perform the combination of $(\mathbf{v}_i^1; \mathbf{v}_i^2; \cdots; \mathbf{v}_i^M)$. We refer this aggregation method as $aggregation_{23}$.

To learn the parameters of the proposed framework LinkedRNN, we need to define a loss function that depends on the specific task. In this work, we investigate LinkedRNN in two tasks – classification and regression.

Classification. The final output of a sequence S^i is $\mathbf{z}_i$. We can consider $\mathbf{z}_i$ as features and build the classifier. In particular, the predicted class labels can be obtained through a softmax function as:

$$p^i = softmax(\mathbf{W}_c \mathbf{z}_i + b_c) \tag{4.8}$$

where $\mathbf{W}_c$ and b_c are the coefficients and the bias parameters, respectively. p^i is the predicted label of the sequence S^i. The corresponding loss function used in this chapter is the crossentropy loss.

Regression. For the regression problem, we choose linear regression in this work. In other words, the regression label of the sequence S^i is predicted as:

$$p^i = \mathbf{W}_r \mathbf{z}_i + b_r \tag{4.9}$$

where $\mathbf{W}_r$ and b_r are the regression coefficients and the bias parameters, respectively. Then square loss is adopted in this work as the loss function as:

$$L = \frac{1}{K} \sum_{i=1}^{K} (y^i - p^i)^2 \tag{4.10}$$

Note that there are other ways to define loss functions for classification and regression. We would like to leave the investigation of other formats of loss functions as one future work.

Prediction. For an unlabeled sequence S^j under the classification problem, its label is predicted as the one corresponding to the entity with the highest probability in $softmax(\mathbf{W}_c \mathbf{z}_j + b_c)$.

For an unlabeled sequence S^j under the regression problem, its label is predicted as $\mathbf{W}_r \mathbf{z}_i + b_r$.

Although the framework is designed for transductive learning, it can be naturally used for inductive learning. For a sequence S^k, which is unseen in the given linked sequences $\mathcal{S}$, according to its sequential information and its neighbors $\mathcal{N}(k)$, it is easy to obtain its representation $\mathbf{z}_k$ via Equation 4.7. Then based on $\mathbf{z}_k$, its label can be predicted as the normal prediction step described above.

Table 4.1: Statistics of the datasets.

Description	DBLP	BOOHEE
# of sequences	47,491	18,229
Network density (‰)	0.13	0.012
Avg length of sequences	6.6	23.5
Max length of sequences	20	29

4.4 Experiment

In this section, we present experimental details to verify the effectiveness of the proposed framework. Specifically, we validate the proposed framework on datasets from two different domains. Next, we firstly describe the datasets we used in the experiments and then compare the performance of the proposed framework with representative baselines. Lastly, we analyze the key components of LinkedRNN.

4.4.1 Datasets

In this study, we collect two types of linked sequences. One is from DBLP where data contains textual sequences of chapters. The other is from a weight loss website BOOHEE where data includes weight sequences of users. Some statistics of the datasets are demonstrated in Table 4.1. Next we introduce more details.

DBLP dataset. We constructed a chapter citation network from the public available DBLP data set[1][111]. This dataset contains information for millions of chapter from a variety of research fields. Specifically, each chapter contains the following relevant information: chapter id, publication venue, the id references of it and abstract. Following the similar practice in [110], we only select chapters from conferences in 10 largest computer science domains including *VCG, ACL, IP, TC, WC, CCS, CVPR, PDS , NIPS, KDD, WWW,*

[1] https://aminer.org/citation.

ICSE, Bioinformatics, TCS. We construct a sequence for each chapter from their abstracts and regard their citation relationships as the sequence relation. Specifically, we first split the abstract into sentences and tokenize each sentence using python NLTK package. Then, we use Word2Vec [89] to embed each word into Euclidean space and for each sentence, we treat the mean of its word vectors as the sentence embedding. Thus, the abstract of each chapter can be represented by a sequence of sentence embeddings. We will conduct the classification task on this dataset, i.e., chapter classification. Thus, the label of each sequence is the corresponding publication venue.

BOOHEE dataset. This dataset is collected from one of the most popular weight management mobile applications, BOOHEE [2]. It contains million of users who self-track their weights and interact with each other in the internal social network provided by the application. Specifically, they can follow friends, make comment to friends' post and mention (@) friends in comments or posts. The recored weights by users form sequences which contain the weight dynamic information and the social networking behaviors result in three networks that correspond to following, commenting, and mentioning interactions, respectively. Previous work [123] has shown a social correlation on the users' weight loss. Thus, we use these social networks as the sequence relation for the weight sequence data. We preprocess the dataset to filter out the sequences from suspicious spam users. Moreover, we change the time granularity of weight sequence from days to weeks to remove the daily fluctuation noise. Specifically, we compute the mean value of all the recorded weights in one week and use it as the weight for that week. For networks, we combine three networks into one by adding them together and filter out weak ties. In this dataset, we will conduct a regression task of weight prediction. We choose the most recent weight in a weight sequence as the weight we aim to predict

[2] https:www.boohee.com

(or the groundtruth of the regression problem). Note that for a user, we remove all social interactions that form after the most recent weight where we want to avoid the issue of using future sequence relation for weight prediction.

4.4.2 Representative Baselines

To validate the effectiveness of the proposed framework, we construct three groups of representative baselines. The first group includes the state-of-the-art network embedding methods, i.e., node2vec [45] and GCN [71], which only capture the sequence relation. The second group is the GRU RNN model [42], which is the basic model we used in our model to capture sequential information. Baselines in the third group is to combine models in the first and second groups, which captures both sequential and sequence relation. Next, we present more details about these baselines.

- Node2vec [45]. Node2vec is one state-of-the-art network embedding method. It learns the representation of sequences only capturing the sequence relation in a random-walk perspective.
- GCN [71] It is the traditional graph convolutional graph algorithm. It is trained with both relation and label information. Hence, it is different from node2vec, which is learnt with only sequence relation and is totally independent on the task.
- RNN [42]. RNNs have been widely used for modeling sequential data and achieved great success in a variety of domains. However, they tend to ignore the correlation between sequences and only focus on sequential information. We construct this baseline to show the importance of correlation information. To make the comparison fair, we employ the same recurrent unit (GRU) in both the proposed framework and this baseline.

- RNN-node2vec. The Node2vec method is able to learn representation from the sequence relation and the RNN can do so from the sequential information. Thus, to obtain the representation of sequences that contains both relational and sequential information, we concatenate the two sets of embeddings obtained from Node2vec and RNN via a feed forward neural network.

- RNN-GCN. RNN-GCN applies a similar strategy of combining RNN and node2vec to combine RNN and GCN.

There are several notes about the baselines. First, node2vec does not use label information and it is unsupervised , RNN and RNN-node2vec utilize label information and they are supervised, and GCN and RNN-GCN use both label information and unlabeled data and they are semi-supervised. Second, some sequences may not have sequence relation and baselines only capture sequence relation cannot learn representations for these sequences; hence, in this work, when representations from sequence relation are unavailable, we will use the representations from the sequential information via RNN instead. Third, we do not choose LSTM and its variants as baselines since our current model is based on GRU and we also can choose LSTM and its variants as the base models.

4.4.3 Experimental Settings

Data split: For both datasets, we randomly select 30% for test. Then we fix the test set and choose $x\%$ of the remaining 70% data for training and $1 - x\%$ for validation to select parameters for baselines and the proposed framework. In this work, we vary x as $\{10, 30, 50, 70\}$.

Parameter selection: In our experiments, we set the dimension of representation vectors

Table 4.2: Performance comparison on the DBLP dataset

Measurement	Method	Training ratio			
		10 %	30 %	50%	70%
Micro-F1	node2vec	0.6641	0.6550	0.6688	0.6691
	GCN	0.7005	0.7093	0.7110	0.7180
	RNN	0.7686	0.7980	0.7978	0.8025
	RNN-node2vec	0.7940	0.8031	0.7933	0.8114
	RNN-GCN	0.7912	0.8230	0.8255	0.8284
	LinkedRNN	0.8146	0.8399	0.8463	0.8531
Macro-F1	node2vec	0.6514	0.6523	0.6513	0.6565
	GCN	0.6874	0.6992	0.7004	0.7095
	RNN	0.7452	0.7751	0.7754	0.7824
	RNN+node2vec	0.7734	0.7797	0.7702	0.7912
	RNN+GCN	0.7642	0.8014	0.8069	0.8104
	LinkedRNN	0.7970	0.8249	0.8331	0.8365

of sequences to 100. For Node2vec, we use the validation data to select the best value for p and q from $\{0.25, 0.50, 1, 2, 4\}$ as suggested by the authors [45] and use the default values for the remaining parameters. In addition, the learning rate for all of the methods are selected through validation set.

Evaluation metrics: Since we will perform classification in the DBLP data, we use Micro and Macro F1 scores as the metrics for DBLP, which are widely used for classification problems [134, 45]. The higher value means better performance. We perform the regression problem weight prediction in the BOOHEE data. Therefore the performance in BOOHEE data is evaluated by mean squared error (MSE) score. The lower value of MSE indicates higher prediction performance.

4.4.4 Experimental Results

We first present the results in DBLP data. The results are shown in Table 4.2. For the proposed framework, we choose $M = 2$ for the link layer and more details about discussions about the choices of its aggregation functions will be discussed in the following section. From the table, we make the following observations:

- As we can see in Table 4.2, in most cases, the performance tends to improve as the number of training samples increases.

- The random guess can obtain 0.1 for both micro-F1 and macro-F1. We note that the network embedding methods perform much better than the random guess, which clearly shows that the sequence relation is indeed helpful for the prediction.

- GCN achieves much better performance than node2vec. As we mentioned before, GCN uses label information and the learnt representations are optimal for the given task. While node2vec learns representations independent on the given task, the representations may be not optimal.

- The RNN approach has higher performance than GCN. Both of them use the label information. This observation suggests that the content and sequential information is very helpful.

- Most of the time, RNN-node2vec and RNN-GCN outperform the individual models. This observation indicates that both sequential and sequence relation are important and they contain complementary information.

- The proposed framework LinkedRNN consistently outperforms baselines. This strongly demonstrates the effectiveness of LinkedRNN. In addition, comparing to RNN-node2vec

Table 4.3: Performance comparison on the BOOHEE dataset

Method	Training ratio			
	10 %	30 %	50%	70%
node2vec	8.8702	8.8517	7.4744	7.0390
GCN	8.9347	8.6830	6.7949	6.7278
RNN	8.6600	8.6048	7.0466	6.8033
RNN-node2vec	8.4653	8.5944	7.0173	6.7796
RNN-GCN	8.6286	8.5662	6.9967	6.7945
LinkedRnn	7.1822	6.3882	6.8416	6.3517

and RNN-GCN, the proposed framework is able to jointly capture the sequential and sequence relation coherently, which leads to significant performance gain.

We present the performance on BOOHEE in Table 4.3. Overall, we make similar observations as these on DBLP as – (1) the performance improves with the increase of number of training samples; (2) the combined models outperform individual ones most of the time and (3) the proposed framework LinkedRNN obtains the best performance.

Via the comparison, we can conclude that both sequential and relation information in the linked sequences are important and they contain complementary information. Meanwhile, the consistent impressive performance of LinkedRNN on datasets from different domains demonstrate its effectiveness in capturing the sequential and relation information presented in the sequences.

4.4.5 Component Analysis

In the proposed framework LinkedRNN, we have investigate several ways to define the two aggregation functions. In this subsection, we investigate the impact of the aggregation functions on the performance of the proposed framework LinkedRNN by defining the following variants.

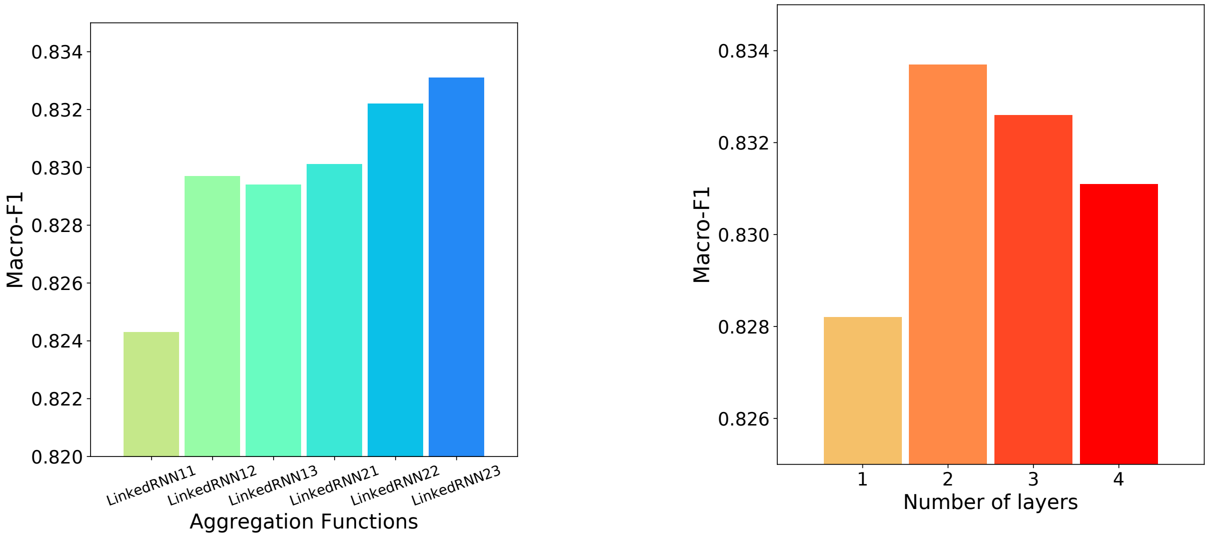
- LinkedRNN11: it is the variant which chooses $aggregation_{11}$ and $aggregation_{21}$
- LinkedRNN12: we define the variant by using $aggregation_{11}$ and $aggregation_{22}$
- LinkedRNN13: this variant is made by applying $aggregation_{11}$ and $aggregation_{23}$
- LinkedRNN21: this variant utilizes $aggregation_{12}$ and $aggregation_{21}$
- LinkedRNN22: it is the variant which chooses $aggregation_{12}$ and $aggregation_{22}$
- LinkedRNN23: we construct the variant by adopting $aggregation_{12}$ and $aggregation_{23}$

The results are demonstrated in Figure 4.3a. Note that we only show results on DBLP with 50% as training since we can have similar observations with other settings. It can be observed:

- Generally, the variants of LinkedRNN with $aggregation_{12}$ obtain better performance than $aggregation_{11}$. It demonstrates that aggregating the sequence of the latent presentations with the help of the attention mechanism can boost the performance.
- Aggregating representations from more layers in the link layer typically can result in better performance.

4.4.6 Parameter Analysis

LinkedRNN uses the link layer to capture sequence relation. The link layer can have multiple layers. In this subsection, we study the impact of the number of layers on the performance of LinkedRNN. The performance changes with the number of layers are shown in Figure 4.3b.



(a) The impact of aggregation functions.

(b) The performance variance with the number of link layers

Figure 4.3: Model analysis results

Similar to the component analysis, we only report the results with one setting in DBLP since we have similar observations. In general, the performance first dramatically increases and then slowly decreases. One layer is not sufficient to capture the sequence relation while more layers may result in overfitting.

4.5 Discussion

RNNs have been proven to be powerful in modeling sequences in many domains. Most of existing RNN methods have been designed for sequences which are assumed to be i.i.d. However, in many real-world applications, sequences are inherently related and linked sequences present both challenges and opportunities to existing RNN methods, which calls for novel RNN methods. In this chapter, we study the problem of designing RNN models for linked sequences. Suggested by Homophily, we introduce a principled method to capture sequence relation and propose a novel RNN framework LinkedRNN, which can jointly model sequential and relation information. Experimental results on datasets from different domains demonstrate that (1) the proposed framework can outperform a variety of representative baselines; and (2) sequence relation is helpful to boost the RNN performance.

There are several interesting directions to investigate in the future. First, our current model focuses on unweighted and undirected relation and we will study weighted and directed links and the corresponding RNN models. Second, in current work, we focus on classification and regression problems with certain loss functions. we will investigate other types of loss functions to learn the parameters of the proposed framework and also investigate more applications of the proposed framework. Third, since our model can be naturally extended for inductive learning, we will further validate the effectiveness of the proposed framework for inductive learning. Finally, in some applications, the sequence relation may be evolving; thus we plan to study RNN models, which can capture the dynamics of links as well.

Chapter 5

Modeling Temporal Information

5.1 Introduction

In many scenarios, sequences are also associated with temporal information. For example, in a customer behavior sequence, each behavior (event) has a time stamp recording the exact time that behavior happened. Such temporal information is essential for recommender systems to make recommendation at the right time, which is vital to improve customers' shopping experience in e-commerce for the following reasons [120, 2, 67, 72, 38]. First, customers' interests in buying a product naturally fluctuate. Thus, there could be right moments when recommending one product will benefit the customer most. On the contrary, there could be huge opportunity cost if the system does it at the moment when the customer loses his/her interest since it may lose great opportunities to attract customers. Second, many products are time-sensitive. For instance, popular snow boots are unlikely well wanted in summer. Therefore, it will greatly impair customers' trust if recommender systems overlook the time information. Third, predicting time about customers' needs will also enable providers to supply enough goods and avoid disappointing customers with “out of order" experience.

The increasing availability of time-stamped fine-grained customers' behaviors observed by providers brings us unprecedented opportunities to build advanced recommender systems that take both customers' preference and its temporal dynamics into consideration. Take a customer who plans to buy a phone for example, she may firstly *search* for some phone brands in e-commerce websites. Then she *clicks* a few models for detailed description and *adds* two of them *to* the *shopping cart* for further comparison. Finally, she *deletes* one in the cart and decides to *purchase* the other one. The fine-grained behaviors as shown in the previous example offer us a new perspective to model customers' preferences. Furthermore, these behaviors are often time-stamped; thus it provides rich temporal information that enables us to gain a deep understanding on temporal dynamics of customers' preferences. Such understanding enables us to perform just-in-time recommendations where we consider what and when to recommend simultaneously.

Although exploiting time-stamped fine-grained users' behaviors will potentially bring in tremendous opportunities to advance recommender systems, new challenges are also introduced. First, considering users' behaviors introduces great complexity to model the relations among various products. The majority of traditional recommender systems merely consider coarse-grained user-item interactions. They often assume that the influence of one product on another is static. However, when considering behaviors users perform on products, such influence could vary according to the specific behavior associated with the product. Below is one example. If a customer clicks “iPhone", it is a strong signal that she may be interested in buying a new phone; thus in addition to presenting iPhone models, we can also recommend some “Samsung" phones to her. In other words, clicking “iPhone" increases the probability of recommending “Samsung" phone. On the other hand, if she just purchased an “iPhone", it is unlikely that she will buy another new phone in the near future. Thus, it is more suitable to recommend “iPhone" cases than “Samsung" phones. In this case, purchasing “iPhone" should decrease the probability of recommending a “Samsung" phone. Therefore, “iPhone"'s influence on “Samsung" phone depends on the behaviors associated such as purchase and click. Second, temporal dynamics of customers' interests are heterogeneously influenced by various products they have interacted with. For example, a customer's interest in buying a new "iPhone" could be related by her previous interactions with other products such as “iPad" and “Samsung" phones, each of which can have different influence according to the behaviors associated. This presents formidable challenges to accurately model temporal dynamics of interests because it requires to consider all the customers' historical interactions, which is further complicated by varied relations among products.

In this chapter, we investigate the just-in-time recommendation problem and address aforementioned challenges simultaneously to leverage the timestamped fine-grained customer behavior data.

5.2 Problem Statement

Let $\mathcal{U} = [u^1, u^2, \cdots, u^N]$ be the set of N users. We assume that $\mathcal{P} = [p^1, p^2, \cdots, p^V]$ is a set of products where V is the number of products. Let $\mathcal{B} = (b^1, b^2, \cdots, b^B)$ be the set of B behaviors users can perform on products such as “click" and “purchase". The historical interactions of a user u^i with the recommender system have been recorded and are represented as a sequence S^i:

$$S^i = \{(t_1^i, b_1^i, p_1^i), (t_2^i, b_2^i, p_2^i), \cdots, (t_k^i, b_k^i, p_k^i)\} \tag{5.1}$$

where each element in S^i is a 3-tuple that is named as an event in this work. Each event $e_j = (t_j^i, b_j^i, p_j^i)$ contains one time stamp t_j^i, one behavior type $b_j^i \in \mathcal{B}$ and one product $p_i^j \in \mathcal{P}$. Moreover, $\mathcal{S} = [S^1, S^2, \cdots, S^N]$ is used to denote the set of event sequences of all users in

$\mathcal{U}$. We further assume that the next event after t_k^i for u_i $(t_{k+1}^i, b_{k+1}^i, p_{k+1}^i)$ is available for all $u_i \in \mathcal{U}$. With above notations, the problem we plan to study in this work can formally defined as:

Given event sequences of a set of users $\mathcal{U}$, i.e., $\mathcal{S}$, and the corresponding next events $\{(t_{k+1}^i, b_{k+1}^i, p_{k+1}^i)\}_{i=1}^N$, we aim to learn a mapping function from $\mathcal{S}$ to $\{(t_{k+1}^i, b_{k+1}^i, p_{k+1}^i)\}_{i=1}^N$; thus it can simultaneously predict next product p_{k+1}^j and its corresponding time stamp t_{k+1}^j given a user $u^j \notin \mathcal{U}$ with its historical event sequence $S^j = \{(t_1^j, b_1^j, p_1^j), (t_2^j, b_2^j, p_2^j), \cdots, (t_k^j, b_k^j, p_k^j)\} \notin \mathcal{S}$.

5.3 The Proposed Framework

In this section, we first give an architecture overview about the proposed framework and then we detail each model component.

5.3.1 An Architecture Overview

In addition to sequential information, sequences also contain temporal information to indicate when events happened, which not only enables us to make the accurate recommendation but also can allow us to perform recommendations at the right time. To achieve this, as mentioned before, major challenges include capturing the sequential information, complex interactions between products and behaviors, and modeling the timing accurately. In this subsection, we will introduce key components of our proposed model that are able to address those challenges and simultaneously recommend right product and predict recommendation time. The overview of our framework is shown in Figure 5.1. It consists of six layers from the bottom to the top: 1) Input layer; 2) Embedding layer, which embeds high-dimensional input

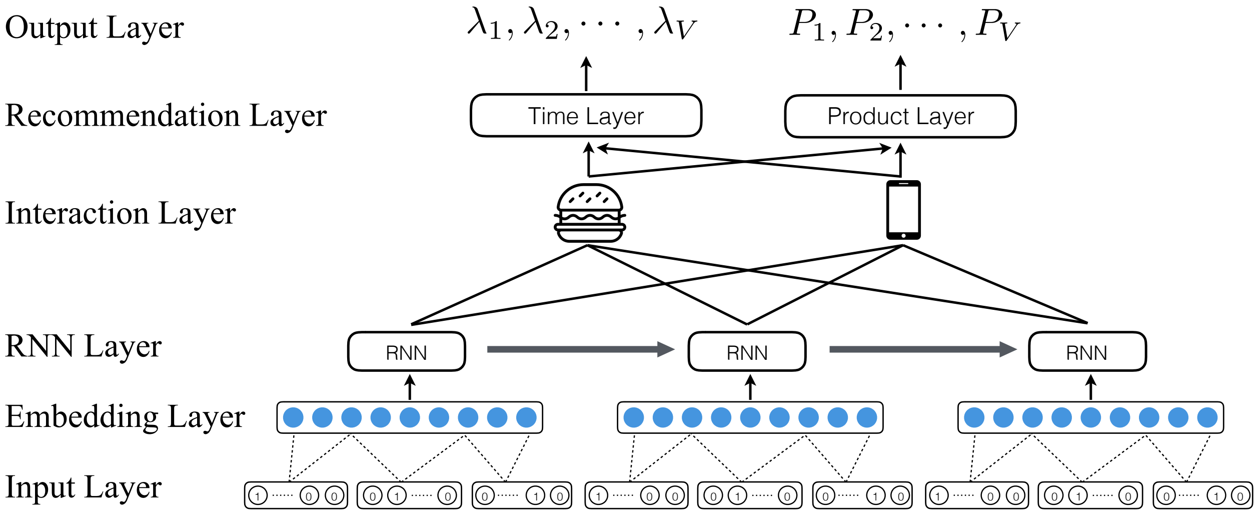


Figure 5.1: The network architecture of the proposed framework. It consists of six key layers: 1) Input layer; 2) Embedding layer, which embeds high-dimensional input vector into low-dimension space; 3) RNN layer, which captures the sequential and temporal information in sequences; 4) Interaction layer, which models the interactions between products and events in the sequence; 5) Recommendation layer, which includes product layer and time layer to predict products and time, respectively; and 6) Output layer where λ_i denotes the time and P_i indicates the probability of recommending product i.

vector into low-dimension space; 3) RNN layer, which captures the sequential and temporal information in sequences; 4) Interaction layer, which models the interactions between products and events in the sequence; 5) Recommendation layer, which includes product layer and time layer to predict products and time, respectively; and 6) Output layer where λ_i characterizes the temporal dynamics of interest in buying i^{th} product and P_i indicates the probability of recommending product i. In the following subsections, we will give details of each layer.

5.3.2 Embedding Layer

The input of the model is the sequences of events, where each event $e = (t, b, p)$ contains timing, behavior and product information. $p = \{0, 1\}^V$ is a one-hot vector and $p[i = j] = 1, p[i \neq j] = 0$ if j^{th} product is involved in e. Similarly, $b = \{0, 1\}^B$ is the one-hot vector for behavior indication. Both products and behaviors space could be very large in real-world recommender systems. For example, there could be millions of products in e-commerce recommender systems. Hence, one-hot vectors are typically high-dimensional, which motivates us to add an embedding layer right after the input layer to embed products and behaviors into low-dimension spaces. Specifically, two embedding matrices $\mathbf{W}^p \in \mathbb{R}^{V \times K^p}$ and $\mathbf{W}^b \in \mathbb{R}^{V \times K^b}$ are introduced. The representation of p and b in the embedding spaces become $p \cdot \mathbf{W}^p$ and $b \cdot \mathbf{W}^b$, respectively. Note that t can also be embedded by using an embedding matrix as it does for b and p. But in this work, to reduce number of parameters, we use binary number of timestamp t to be the time embedding t^e and pad zeros to the left to make all of time embedding vectors have the same length. Thus, each event e is the concatenation of behavior, product and temporal information in their embedding spaces as:

$$e = [p \cdot \mathbf{W}^p \| b \cdot \mathbf{W}^b \| t^e] \tag{5.2}$$

where $\|$ is the concatenation operator. As $K^p \lll V$ and $K^b \lll B$, the resulted representation of events is dense, which will be the input into the following RNN layer to capture the sequential information of events.

5.3.3 RNN Layer: Learning Sequential Information

The order and time of user behaviors in the event sequence should carry important information to understand user's interest. In this subction, we design a RNN sublayer to effectively model the sequences. Specifically, at time-step t, the RNN receives an input vector $\mathbf{e}_t \in \mathbb{R}^E$, involving behavior $b_t \in \mathcal{B}$, product $p_t \in \mathcal{P}$ and a latent representation $\mathbf{h}_{t-1}$ of the sequence seen previously. It emits an output vector $\mathbf{h}_t$, an updated representation that incorporates the new observation from $\mathbf{e}_t$. Due to its simplicity and performance, we choose GRU as the RNN cell. The RNN sublayer will output hidden states $\mathbf{h}_1, \mathbf{h}_2, \cdots, \mathbf{h}_t$ with $\mathbf{h}_i$ representing the sequential information up to time stamp t_i.

5.3.4 Interaction Layer: Modeling Heterogeneous Influence

Although, the hidden representation $\mathbf{h}_t$ produced by GRU based RNN contains sequential information of events in a sequence and has been successfully utilized for recommendation [56]. However, GRU only models the sequence and fails to consider how the events in sequences will affect later decision on each product. In fact, later interest of a product is highly related to past behaviors with products. For example, if a user clicks one cellphone several times in the near past, it is very likely that she has interest in buying a new phone. Thus, it is desirable to recommend similar products for her. In another case, if the user has already purchased a cellphone, the probability that she will purchase it again in the near future is very low. Thus, instead of similar products, related products such as phone cases are likely to match her interests. However, for other types of products such as food, the story could be totally different. For instance, if the user has purchased popular and positively rated snacks, it is highly possible that she will buy it again. In this case, previous behaviors increase the probability of the same behaviors. Thus, later interest of a product is complicatedly related to both previous behaviors and products in the sequence. To capture the heterogeneous influence of events on different products, we learn a unique latent representation $\mathbf{c}^i$ of the sequences for each product. To be specific, with the sequence of hidden representation $\mathcal{H} = \{h_1, h_2, \cdots h_k\}$ produced by previously introduced RNN layer and the candidate product embedding vectors $\mathbf{W^p_{1,:}}, \mathbf{W^p_{2,:}}, \cdots, \mathbf{W^p_{V,:}}$, we compute the relation between the j^{th} product's embedding $\mathbf{W^p_{j,:}}$ and each hidden representation $\mathbf{h}_i$ in $\mathcal{H}$ through a function $a(\cdot)$ followed by a Softmax function as follows:

$$\alpha_{ij} = \frac{e^{a(\mathbf{h}_i, \mathbf{W}^{\mathbf{p}}_{\mathbf{j},:})}}{\sum_k e^{a(\mathbf{h}_k, \mathbf{W}^{\mathbf{p}}_{\mathbf{j},:})}} \tag{5.3}$$

where $a(\cdot)$ could be any functions such as feedforward neural networks and dot product. As the dot product is computed much faster practically due to the highly optimized matrix multiplication code [117], in this work, we have chosen it as the function $a(\cdot)$ such that $a(\mathbf{p}, \mathbf{q}) = \mathbf{p}^T\mathbf{q}$. The output latent vector $\mathbf{c}_j$ which is the unique representation of the sequence history on the j^{th} product can be then computed by the weighted sum of hidden representation vectors as follows:

$$\mathbf{c}_j = \sum_i \alpha_{ij}\mathbf{h}_i \tag{5.4}$$

Thus, the heterogeneous influence of the events in a sequence on a product j is captured and contained in the unique representation vectors $\mathbf{c}_j$. With $\mathbf{c}_j$, we are able to design product and time layers for product recommendation and time prediction.

5.3.5 Product Layer: Recommending the Product

For the j^{th} product, previous layer outputs a unique vector $\mathbf{c}_j$, which captures both sequential information and relations between the j^{th} product and each event in the sequence and represents the user's later interest on that product. In this subsection, we describe the product layer that utilizes $\mathbf{c}_j$ and embedding vector of j^{th} product $\mathbf{W}^p_{j,:}$ to make recommendations. Specifically, the product layer involves a non-linear function $s(\cdot)$, which synthesizes $\mathbf{c}_j$ and $\mathbf{W}^p_{j,:}$ to obtain a score which indicates user's future interest on the j^{th} product. Due to the computation efficiency mentioned in the previous subsection, we also choose $s(\cdot)$ to be the dot product such that $s(\mathbf{p} \cdot \mathbf{q}) = \mathbf{p}^T\mathbf{q}$. Note that other functions such as feed-forward neural networks can also be used, we leave it as one future exploration direction. To obtain interest probability distribution over all the candidate products, we apply a softmax layer on the top of $S(\cdot)$:

$$P(\tilde{y} = i^{th} product) = \frac{\exp(s(\mathbf{W}^p_{i,:}, \mathbf{c}^i))}{\sum_j \exp(s(\mathbf{W}^p_{j,:}, \mathbf{c}^j))} \tag{5.5}$$

Given the probability distribution, we can rank products according to their probabilities for recommendations.

5.3.6 Time layer: Predicting Recommendation Time

Beside sequential information, event sequences also provide rich temporal information that can be leveraged to model user's interest dynamics to make the recommendation just in time. For example, a user's interest of a certain product will naturally fluctuate instead of staying the same. Accurately modeling these dynamics enables us recommend the right product at the time when user needs it. However, as mentioned previously, it is very challenging to model the dynamics of user interest as it varies from product to product. Fortunately, the unique representation $\mathbf{c}_j$ of users history information for each product enables us to achieve the goal. In this subsection, we will describe our time prediction layer that leverages $\mathbf{c}_j$ to model the temporal dynamics of user's interest for j^{th} product. Next, we briefly introduce temporal point process to model temporal dynamics.

Temporal point process is a random process whose realization consists of a sequence of isolated events with a time stamp t for each event and is denoted by $\{t_i | i = 1, 2, \cdots, N\}$, where N is the number of events and $t_i \in \mathbb{N}^+$. Note that temporal point process is also called counting process in some literature and the two terminologies are interchangeable. In this work, we will stick to the former one. A point process is often modeled by its conditional intensity function $\lambda(t|\mathcal{H}_t)$, where $\mathcal{H}_t$ is the history before t, which represents the rate of events happening at time t given the time stamps of all events happening before t. The formal definition of $\lambda(t|\mathcal{H}_t)$ is the following [1]:

$$\lambda(t|\mathcal{H}_t) = \lim_{\Delta t \to 0} \frac{1}{\Delta t} \frac{S(t|\mathcal{H}_t) - S(t + \Delta_t|\mathcal{H}_t)}{S(t|\mathcal{H}_t)} \tag{5.6}$$

where $S(t|\mathcal{H}_t)$ is the conditional probability that the new event will not happen before t given $\mathcal{H}_t$. With Equation 5.6, $S(t|\mathcal{H}_t)$ can be straightforwardly written as:

$$S(t'|\mathcal{H}_t) = \exp\left\{ -\int_{t'}^{t} \lambda(\tau|\mathcal{H}_t) d\tau \right\} \tag{5.7}$$

Thus, the conditional probability density of an event happening at time stamp t_i is $f(t_i) = \lambda(t_i)S(t_i)$ and the likelihood of the realization $\{t_i|i = 1, 2, \cdots, N\}$ is:

$$\mathcal{L}(\{t_i|i = 1, 2, \cdots, N\}) = \sum_{i}^{n} \log \lambda(t_i) - \int_{0}^{t_n} \lambda(\tau) d\tau \tag{5.8}$$

Depending on the assumptions about parametric form of $\lambda(t|\mathcal{H}_t)$, the temporal point process can be reduced to more specific ones. For example, if $\lambda(t|\mathcal{H}_t) = \lambda(t) = \lambda_0$, it is the Poisson process [70]. A more general point process is inhomogeneous Poisson process where $\lambda(t|\mathcal{H}_t) = \lambda(t)$. Moreover, if $\lambda(t)$ increases or decreases when new event comes, the temporal point process becomes Hawkes [52] or self-correcting process [62], respectively. Although, these specific temporal point processes whose intensity function $\lambda(t)$ takes assumed parametric form has shown it effectiveness in various applications [75, 50, 91], it requires prior domain knowledge, which is not always available, to make a reasonable choice of the form. In addition, this approach is constrained by the chosen parametric forms because it may not align well with the actual data, which is generated by a much more complex underlying process. To leverage the effectiveness of temporal point process in modeling event sequences and avoid issues with parametric form of intensity function, in the next, we exploit the high-level representation that contains the temporal information of the sequences to model the intensity function.

Modeling intensity function: The interaction layer outputs a unique high-level representation of the temporal information of user history for each product. Thus, it is natural to model the intensity function that characterizes the underlying dynamics of user's interest on a product with its corresponding representation vector. Specifically, the intensity function for the future interest on j^{th} product can be learned as follows:

$$\lambda^j(\delta t) = \sigma(\mathbf{w}_t \cdot \mathbf{c}^j) e^{-\sigma(\mathbf{w}_d \cdot \mathbf{c}^j)\delta t} \tag{5.9}$$

where δt is the elapsed time since the last event in the sequence happening, $\mathbf{w}_d$ and $\mathbf{w}_t$ are the model learnable parameters, $\sigma(\mathbf{w}_t \cdot \mathbf{c}^j)$ determines how much previous events trigger the intensity for j^{th} product and $\sigma(\mathbf{w}_d \cdot \mathbf{c}^j)$ determines how quickly the influence diminishes.

Predicting the time of next event: With the intensity function $\lambda^j(\delta t)$, the probability of event involving the j^{th} product happening at time t is:

$$\begin{aligned} p^j(t) =& \lambda^j(t)\big(- \int_0^t \lambda^j(\tau) d\tau\big) \qquad (5.10) \\ =& \sigma(\mathbf{w}_t \cdot \mathbf{c}^j) \exp\Big(- \sigma(\mathbf{w}_d \cdot \mathbf{c}^j)\delta t \end{aligned}$$

$$+\frac{\sigma(\mathbf{w}_t \cdot \mathbf{c}^j)}{\sigma(\mathbf{w}_d \cdot \mathbf{c}^j)} \exp(-\sigma(\mathbf{w}_d \cdot \mathbf{c}^j)\delta t - 1)\Big)$$

To predict the time of event involving j^{th} product happening, we calculate the expected value of $p^j(t)$ as follows:

$$\hat{t}^j = \int_0^\infty t \cdot p^j(t)dt \tag{5.11}$$

Thus, $\hat{t}^j$ indicates the right time to recommend the j^{th} product.

5.3.7 Learning Model Parameters

Given $\mathcal{S}$ and their next event $\{(t_{k+1}^i, b_{k+1}^i, p_{k+1}^i)\}_{i=1}^N$, one straightforward loss function of the proposed framework to learn model parameters is the negative sum of log likelihood for events:

$$\mathcal{L} = \sum_{i=1}^{N}(\beta\mathcal{L}_i^p + (1-\beta)\mathcal{L}_i^t) \tag{5.12}$$

where $\mathcal{L}_i^t = -\log P^{p^i_{k+1}}(t_{k+1}^i)$ and $\mathcal{L}_i^p = -\log P(\tilde{y} = p_{k+1}^i)$ are the loss of temporal and product modeling for i^{th} sequence, respectively. And β controls the relative emphasis of time or item prediction accuracy, N is the number of total sequences, p_{k+1}^i and t_{k+1}^i are the product and time in the next event of i^{th} sequence, respectively.

However, we empirically found that the above loss function could lead to very unstable performance which is consistent with other works [56]. To address this, we adopt negative sampling strategy in the training process. Specifically, for each positive product p_{k+1}^i, we randomly sample one negative product p_{neg}^i from $\mathcal{P}$ and enforce $P(\tilde{y} = p_{k+1}^i) > P(\tilde{y} = p_{neg}^i)$

by the logits loss for binary classification as follows:

$$\mathcal{L}_i^p = -\log \sigma(s(\mathbf{W}^p_{pos,:}, \mathbf{c}^{pos})) - \log(1 - \sigma(s(\mathbf{W}^p_{neg,:}, \mathbf{c}^{neg})) \tag{5.13}$$

where $s(\cdot)$ is the dot product function used in Equation 5.5, and pos and neg are the product indexes of p^i_{pos} and p^i_{neg}, respectively.

5.4 Experiment

In this section, we conduct extensive experiments with real-world data to evaluate the proposed model JRec. We first describe details of the data followed by experimental settings. Then we compare the performance of JRec in the tasks of product recommendation and event time prediction with that of representative baselines.

5.4.1 Experimental Settings

Data Description: The data used in experiments is collected from a popular e-commerce website JD.com. This data records click and purchase events from customers' interactions with the website. Thus, each event has two pieces of information: one the user behavior type (click or purchase); the other is the product. We collect $6,166,916$ sessions and these sessions contain $169,856$ products. To avoid data sparsity problem, we filter out sequences that contain products that appear less than one hundred times in the data. We further filter out very short sequences in order to analyze how the length of the sequence affect the performance. We follow the procedure in [56] and split the data into training, validation and testing sets. Each sequence will be in one and only one of the three sets.

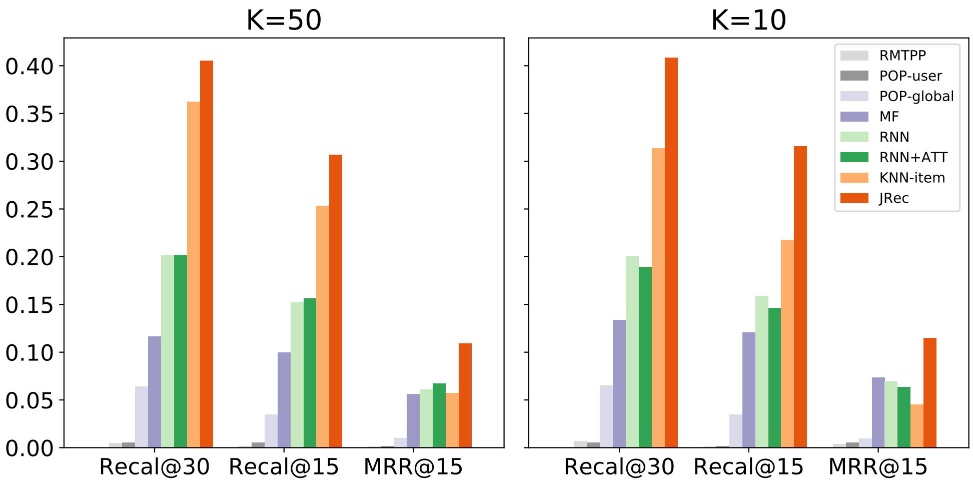


Figure 5.2: Performance comparison on next item prediction. The prediction performance is measured by Recall@30, Recal@15 and MRR@15. The left and right panels show the results with long and short sequences, respectively. It is easily observed that the proposed framework JRec achieves the best performance with any metrics.

Experiment settings: To analyze how the length of event sequence will affect the model performance, we further construct two datasets from the original data. Specifically, we cut the last 10 and 50 events of the training sequences to make one of the datasets only contain the sequences of length 10 and the other only contain those of 50. This also ensures that both datasets have the same set of sessions, so that we are able to control variables other than sequence length that could potentially affect the prediction performance.

5.4.2 Performance On Next Item Recommendation

In this subsection, we evaluate our proposed model in terms of its accuracy in predicting customers' next interested product. Our model will produce a probability distribution over all the products. To consider real-world scenarios where a recommender system often recommends a couple of products that match customers' interest most, we choose two metrics to measure the model performance. The first metric is Recall@x, which is the proportion of cases that the desired products are ranked in top x in terms of probability. We vary the value of x such that $x = \{15, 30\}$. The second metric used for evaluation is Mean Reciprocal Rank (MRR), which is the average of the reciprocal probability ranks of the desired products. Thus, in contrast to Recall@x that measures system general recommendation performance, MRR focuses on the system's ability to recommend user the most desired product. These two metrics are widely used in previous works to evaluate the effectiveness of a recommender system and their mathematical expressions are shown as follows:

$$Recall@x = \frac{1}{N}\sum_i I(rank_i < x) \tag{5.14}$$

$$MRR = \frac{1}{N}\sum_i \frac{1}{rank_i}$$

where $I(x) = 1$ is x is true and $I(x) = 0$ otherwise. The higher Recall@x and MRR are, the better the performance is.

To show the recommendation performance of the proposed model, we compare it with the following representative baselines:

POP-global: This baseline recommends the most popular products in the training data to customers.

POP-user: Instead of a global view, this method considers customer-specific information and recommends products that have been most popular in the given customer's history.

Matrix factorization(MF): MF embeds products and customers into low-dimension spaces according to the customer-item interactions and it is a widely adopted approach in recommender systems. However, in session-based settings, MF can not be directly applied as the customer set is not fixed and there could be new one to come. Thus, we follow the approach in [56] that when a new session comes, we compute the average embeddings of all the products in the session history and use it as the embedding of session. To obtain the product embeddings, we construct and factorize an interaction matrix in which the value in i^{th} row and j^{th} column is the number of occurrence of the j^{th} product in the i^{th} session.

RNN: Recurrent Neural Network is one of most effective methods in modeling sequential information and has gained great success in diverse tasks. Recently it is also adopted in recommender systems and achieves the state-of-the-art performance. In this baseline, we use the model described in [56] and choose the GRU as the recurrent unit to make the comparison fair. Note that this baseline is also known as GRU4REC [56] and has been widely used as baselines from session recommendation tasks.

RNN+ATT: This baseline is a variant of the RNN model. It adds one attention layer on top of the basic GRU model. As attention mechanism is able to identify important information in a sequence, comparing to RNN model, this baseline will demonstrate the effectiveness of attention mechanism.

RMTPP: It models event data based on recurrent neural network and marked temporal point process. This baseline is able to simultaneously predict the type and time of next event [33]. Thus, if we regard the product as the event type, it can be naturally used to jointly predict next product and time.

Implementation Details: All the deep methods are implemented in Pytorch [1] and we optimize them by mini-batch stochastic Adam algorithm [69]. For RMTPP, we directly use the code provided by the authors [2]. In addition, to make the comparison fair, in this work, we use the same network settings (i.e. number of layer, hidden size, mini-batch size etc.) to all deep methods and we select learning rate from $[0.1, 1e^{-1}, 1e^{-2}, 1e^{-3}, 1e^{-4}]$ according to the validation performance.

[1] http://pytorch.org/

[2] https://github.com/musically-ut/tf_rmtpp

Results: The results of recommendation performance of all methods are shown in Figure 5.2. Following observations can be made from the figure. (1) All model based methods achieved better performance than heuristics which recommend products merely based on their popularity. (2) RNN-based methods that are able to capture the sequential information have substantial performance gain over the MF methods, which is consistent with the previous observations [56]. (3) Comparing to RNN, RNN+ATT performs better with long sequences and comparable with short ones. This can be explained that it is hard to capture the long-term dependencies in long sequences and attention mechanism can mitigate the problem by making the model focus on related events. In contrast, it is much easier for RNN to capture the dependencies in short sequences and attention mechanism may not be so useful for short sequences. (4) RMTPP obtains worst performance. We suspect this is because RMTPP is designed for the case where there are only limited number of event types, e.g, less than 10. But the number of event types in our task is much larger as there are huge amount of items in the e-commerce scenario. (5) JRec significantly outperforms all baselines. Specifically, Table 5.1 shows the performance improvement of JRec compared to the best performance of the baselines. We argue that JRec is able to make much better recommendation because of two key advantages. One is the RNN layer that is able to capture sequential information in the sequences. The other, which is more important, is the interaction layer that can exploit the complex relation among behaviors and products. For instance, it is able to attend to the highly relevant events, which will increase the customer's interest on that certain product.

To summarize, the proposed framework is able to outperform all the baseline methods because (1) it models sequential and temporal information; (2) it effectively captures the complex relations between products when behaviors present; and (3) it explicitly models the varied influence of past events on different products.

Table 5.1: Performance improvement of JRec compared to the best performance of the baselines (%)

Sequence length	Recall@30	Recall@15	MRR@15
K=50	+11.86	+21.08	+62.07
K=10	+30.28	+44.81	+55.90

5.4.3 Performance On Next Time Prediction

In this subsection, we will focus on evaluating the performance of our model on time prediction. Specifically, given the product in the next event, we will show how accurately our model can model customers' interest temporal dynamics by predicting when the next event will happen. This is a key task for recommender systems that aim to make just-in-time recommendation.

Evaluation: We use Mean Absolute Error (MAE) as the accuracy measurement, which is computed as: $MAE = \frac{1}{N}\sum_i(|t_i - \hat{t_i}|)$, where t_i and $\hat{t_i}$ are the predicted and actual time of next event of the i^{th} sequence, respectively. A lower MAE score means better performance.

We compare the proposed model with the following representative methods for next time prediction:

Mean-global: This method predicts the next time based on the average time intervals in the training data.

Mean-local: Rather than focusing on global information, this approach only uses the local information and predicts the next time of the session by computing the average of historical time intervals in the session.

Hawkes: Hawkes process model is widely used to model event data with time stamps. It is based on the “self-excited" assumption that the advent of event will increase the rate of new event coming.

RMTPP: As noted in the previous subsection, this method models the event type and time simultaneously and can directly predict the time of the next event [33].

Results: The results for time prediction task are shown in Table 5.2. From the table, we observe that: 1) The Hawkes method obtained the worst performance. We argue this comes from the incorrect assumption that the underlying process that generates the click-purchase data is self-excited. In fact, the advent of one event may even prohibit the happening of next event. For instance, after purchasing a very expensive product, a customer may use up most of her budget and will not likely shop anything for some time. While Hawkes process has achieved great success in modeling event data, its performance can drop substantially if the assumption is not satisfied. Thus, this gives an alarm of applying parametric models which largely depends on the correctness of the assumption underlying the data. 2) RMTPP significantly outperforms both heuristics and Hawkes. This is because RMTPP can utilize RNNs to effectively capture the underlying mechanism generating the sequences. Unlike its bad performance for product prediction, the temporal modeling challenge in datasets is very similar to the case for which the model is designed. Thus, it is able to obtain state-of-the-art performance. 3) JRec has the best performance among all methods with both long and short sequences. It also outperforms RMTPP, which is designed for event time prediction. we argue this is because the JRec is able to capture the varied influence of the historical events on individual product.

5.5 Discussion

In this chapter, we study just-in-time recommendation problem that is to recommend a right product to a customer at the right time. Specifically, we propose a novel model JRec that leverages the new opportunities brought by the fine-grained temporal behavior data and

Table 5.2: Next event time prediction performance comparison.

Baselines	K=50 MAE	K=10 MAE
Mean-global	1038.96	1107.04
Mean-user	1050.38	1050.38
Hawkess	1247.36	3476.80
RMTPP	890.09	892.80
JRec	887.01	889.15

simultaneously predicts the probability of customers' interest and its temporal dynamics. Specifically, JRec not only models the sequential information of the event sequences, but also captures the varied influence of sessions historical data on each product and obtains unique representations, which pave a way for product prediction and interest dynamics modeling with temporal point process. Extensive experiments with real-world e-commerce data have shown that JRec is able to outperform representative baselines for both next product and time prediction.

In this chapter, we focus on session-based recommendation setting and treat each session independently. Thus, one interesting future direction is to extend current framework to capture the relation among customers', where event sequences are related to others. Moreover, we only consider customers' interest for products in general and ignore that of a behavior type. For example, for music service providers, knowing a customer will listen a song or buy a song could be very important for their marketing strategies. Thus, we would also like to extend our framework to predict product, time and behavior type simultaneously.

Chapter 6

Modeling Hierarchy Information

6.1 Introduction

Most of the widely used language processing systems have been built on neural networks that are highly effective, achieving the performance comparable to humans [28, 132, 135]. They are also very brittle, however, as they could be easily broken with the presence of noises [12, 140, 34]. However, the language processing mechanism of humans are very robust. One representative example is the following *Cambridge* sentence:

> *Aoccdrnig to a rscheearch at Cmabrigde Uinervtisy, it deosn't mttaer in waht oredr the ltteers in a wrod are, the olny iprmoetnt tihng is taht the frist and lsat ltteer be at the rghit pclae. The rset can be a toatl mses and you can sitll raed it wouthit porbelm. Tihs is bcuseae the huamn mnid deos not raed ervey lteter by istlef, but the wrod as a wlohe.*

In spite of the fact that a human can read the above sentence with little difficulty, it can cause a complete failure to existing natural language processing systems such as Google Translation Engine [1]. Building robust natural language processing systems is becoming increasingly important nowadays given severe consequences that can be made by adversarial samples [49]: carefully misspelled spam emails that fool spam detection systems [37] deliberately designed input sentences that force chatbot to emit offensive language [126, 29, 81],

[1] https://translate.google.com/

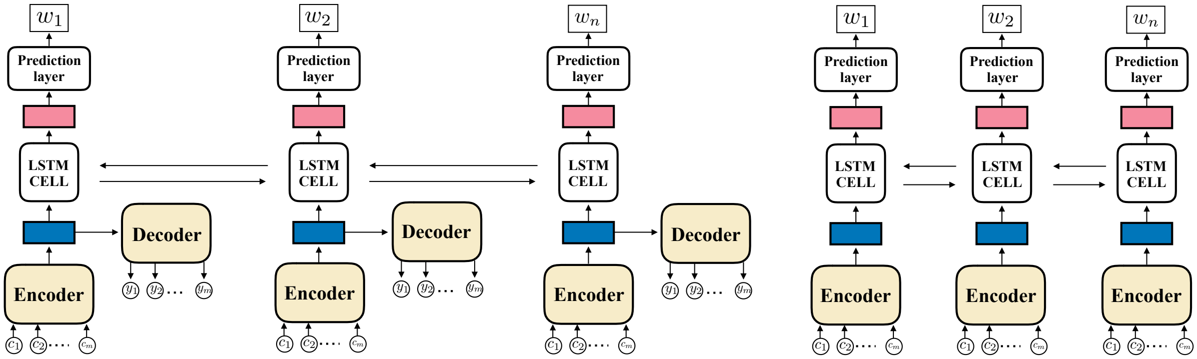


Figure 6.1: The graphical illustration of the proposed framework: MUDE.

etc. Thus, in this work, we focus on building a word recognition framework which can denoise the misspellings such as those shown in the *Cambridge* sentence. As suggested by psycholinguistic studies [99, 27], the humans can comprehend text that is noised by jumbling internal characters while leaving the first and last characters of a word unchanged. Thus, an ideal word recognition model is expected to emulate robustness of human language processing mechanism.

The benefits of such framework are two-folds. The first is its recognition ability can be straightforwardly used to correct misspellings. The second is its contribution to the robustness of other natural language processing systems. By serving as a denoising component, the word recognition framework can firstly clean the noised sentences before they are inputted into other natural language processing systems [98, 144].

From the human perspective, there are two types of information that play an essential role for us to recognize the noised words [96]. The first is the character-level dependencies. Take the word '*wlohe*' in the *Cambridge* sentences as an example, it is extremely rare to see a 'w' sits next to an 'l' in an English word. Instead, it is more natural with 'wh'. Thus, it is quite easy for humans to narrow down possible correct forms of '*wlohe*' to be '*whole*' or '*whelo*'. To ensure that it should be '*whole*', we often need the second type of information: context information such as '*but the wrod as a wlohe.*', which is denoted as word-level dependencies in this chapter. The character-level and world-level dependencies form hierarchical structures. Intuitively, an effective word recognition framework should capture these hierarchical structures. However, hierarchical structures are rarely exploited by the exiting works such as scRNN [102]. Hence, we propose a framework MUDE that is able to fully utilize hierarchical structures for the robust word recognition task. It integrates a character-level sequence-to-sequence model and a word-level sequential learning model into a coherent framework. The major contributions of our work are summarized as follows:

- We identify importance of hierarchical structures for recognizing a noised word;
- We propose a novel framework, MUDE, that utilizes both character-level and word-level dependencies for robust word recognition task;
- We conduct extensive experiments on various types of noises to verify the effectiveness of MUDE.

6.2 The Proposed Framework: MUDE

In this section, we describe MUDE that is able to capture both character-level and word-level dependencies. The overall architecture is illustrated in Figure 6.1. It consists of 3 major components: a sequence-to-sequence model, a bi-directional recurrent neural network and a prediction layer. Next we will detail each component.

6.2.1 Learning Character-level Dependencies

As mentioned previously, there exist sequential patterns in the characters of a word. For example, vocabulary roots such as *cur* and *bio* can be found in many words. In this subsection, we propose a sequence-to-sequence model to learn a better representation of a given word by incorporating character-level dependencies. The model consists of an encoder and a decoder, which we will describe next.

6.2.1.1 Encoder

Let $\hat{w} = c_1, c_2, \cdots c_m$ be a sequence of characters of a given noised word $\hat{w}$. We firstly map each character c_i to a d_c-dimensional character embedding as follows:

$$x_i = \mathbf{E} o_i \tag{6.1}$$

where $\mathbf{E} \in \mathbb{R}^{C \times d_c}$ is the embedding matrix given that the total number of unique characters is C. $o_i \in \mathbb{R}^C$ is the one-hot representation of c_i. Since there could some noise in $\hat{w}$, the sequential order of c_i can be misleading. Thus, instead of using a sequential learning model such as recurrent neural network, we choose the multi-head attention mechanism [117] to model the dependencies between characters without considering their order. To do so, we add a special character c_0 whose final representation will be used as the representation of the word.

Specifically, the multi-head attention mechanism will obtain a refined representation for each character in $\hat{w}$. Next, without the loss of generality, we will use c_i as an example to illustrate. To obtain the refined representation of c_i, x_i will firstly be projected into query space and $x_j \; \forall j \in \{0, 1, \cdots, m\}$ will be projected into key and value spaces as follows:

$$\begin{aligned} x_i^q &= \mathbf{Q} x_i \\ x_j^k &= \mathbf{K} x_j \quad \forall j \in \{0, 1, \cdots, m\} \\ x_j^v &= \mathbf{V} x_j \quad \forall j \in \{0, 1, \cdots, m\} \end{aligned} \tag{6.2}$$

where $\mathbf{Q}$, $\mathbf{K}$, $\mathbf{V}$ are the projection matrices for query, key, and value spaces, respectively. With x_i^q, x_j^k and x_j^v, the refined representation e_i of c_i can be calculated as the weighted sum of x_j^v:

$$e_i = \sum \alpha_j x_j^v \tag{6.3}$$

where α_j is the attention score that is obtained by the following equation:

$$\alpha_0, \alpha_1, \cdots, \alpha_m = \sigma^s\left(\frac{x_i^{qT} x_0^k}{\sqrt{d}}, \frac{x_i^{qT} x_1^k}{\sqrt{d}}, \cdots, \frac{x_i^{qT} x_m^k}{\sqrt{d}}\right)$$

Where σ^s is the softmax function. To capture the dependencies of characters from different aspects, multiple sets of projection matrices are usually used, which will result in multiple sets of x_i^q, x_j^k and x_j^v, and thus e^i. To be concrete, assume that there are h sets of projection matrices, from Equation 6.2 and Equation 6.3, we can obtain h e_is, which are denoted as $\{e_i^1, e_i^2, \cdots, e_i^h\}$. With this, the refined representation of c_i is obtained by the concatenation operation:

$$z_i = concatenation(e_i^1, e_i^2, \cdots, e_i^h) \tag{6.4}$$

where z_i is the new representation of c_i and contains dependency information of c_i to other characters in $\hat{w}$ from h aspects.

Following [117], we also add a positional-wise feedforward layer to z_i as follows:

$$p_i = \mathbf{W}^2 ReLU(\mathbf{W}^1 z_i) \tag{6.5}$$

where $\mathbf{W}^1$ and $\mathbf{W}^2$ are the learnable parameters. p_i is the final representation of c_i. Note that we can have several above mentioned layers stacked together to form a deep structure.

At this point, we have obtained the refined representation vector for each character and we use that of the special character c_0 as the representation of given noised word, which is denoted as w^c

6.2.1.2 Decoder

To capture the sequential dependency in the correct words, the Gated Recurrent Unit (GRU) which has achieved great performance in many sequence learning tasks [129, 6, 128] is used as the decoder. To be specific, in the decoding process, the initial hidden state h_0 of GRU is initialized with the noised word presentation $\hat{w}$. Then at each time stamp t, GRU will recursively output a hidden state h_t given the hidden state h_{t-1} at the previous time stamp. Due to the page limitation, we do not show the details of GRU, which is well described in [20]. In addition, each hidden state will emit a predicted character c_t^p. The decoding process will end when the special character denoting the end of word is emitted. Concretely, the whole decoding process is formally formulated as follows:

$$h_0 = w^c \tag{6.6}$$

$$h_t = GRU(h_{t-1})$$

$$p_t = \sigma^s(\mathbf{W}^p h_t)$$

$$c_t^p = \arg\max_i(p_t[i])$$

where $\mathbf{W}^p \in \mathbb{R}^{C \times d}$ is a trainable parameter. $p_t \in \mathbb{R}^C$ gives the emission probability of each character and $p_t[i]$ denotes the i^{th} entry of vector p_t.

6.2.1.3 Sequence-to-sequence Loss

To train the previously described character-level sequence-to-sequence model, we define the loss function as follows:

$$\mathcal{L}_{seq2seq} = -\sum_{i}^{m} p_i[y_i] \tag{6.7}$$

where y_i is the index of the ground truth at position i of the correct word w. By minimizing $\mathcal{L}_{seq2seq}$, the designed sequence-to-sequence model can learn a meaningful representation that incorporates character-level sequential dependencies for the noised word. Next, we will describe the framework component that captures the word-level dependencies.

6.2.2 Capturing Word-level Dependencies

From the human perspective, it is vitally important to consider the context of the whole sentences in order to understand a noised word. For example, it would be very hard to know 'frist' means 'first' until a context 'the olny iprmoetnt tihng is taht the frist and lsat ltteer be at the rghit pclae.' is given. Thus, to utilize the context information and word-level dependencies, we design a recurrent neural network (RNN) to incorporate them in the noised word representation. Specifically, the word presentations obtained from character-level encoder will be passed into a bi-directional long short-term memory (LSTM). Concretely, given a sequence of word presentations $S = \{w_1^c, w_2^c, \cdots, w_n^c\}$ obtained from character-level dependencies, we calculate a sequence of refined word representation vectors $\{w_1, w_2, \cdots, w_n\}$ as follows:

$$\begin{aligned} w_1^f, w_2^f, \cdots, w_n^f &= LSTM_{forward}(w_1^c, w_2^c, \cdots, w_n^c) \\ w_1^b, w_2^b, \cdots, w_n^b &= LSTM_{backward}(w_1^c, w_2^c, \cdots, w_n^c) \\ w_1, w_2, \cdots, w_n &= w_1^f || w_1^b, w_2^f || w_2^b, \cdots, w_n^f || w_n^b \end{aligned} \tag{6.8}$$

where $\|$ denotes concatenation. $LSTM_{forward}$ indicates that w^cs are processed from w_1^c to w_n^c, while $LSTM_{backward}$ processes word presentations in an opposite direction, namely, from w_n^c to w_1^c. Comparing to original LSTM where only forward pass is performed, bi-directional LSTM can include both 'past' and 'future' information in the representation of w_i.

With the aforementioned procedure, the representation of each word now incorporates both character-level and word-level dependencies. Thus, the correct word is predicted as follows:

$$\begin{aligned} p_i^w &= \sigma^s(\mathbf{W}^w w_i) \\ w_i^p &= \arg\max_i(p_t^w[i]) \end{aligned} \tag{6.9}$$

where $\mathbf{W}^w \in \mathbb{R}^{V \times d_w}$ is a trainable matrix and V is the size of the vocabulary that contains all possible words. Moreover, $p_i^w \in \mathbb{R}^V$ is the probability distribution over the vocabulary for the i^{th} word in a sentence.

6.2.2.1 Word Prediction Loss

To effectively train MUDE for correct word prediction, similar to character-level sequence-to-sequence model, we define the following objective function:

$$\mathcal{L}_{pred} = -\sum_{i}^{n} p_i^w[y_i^w] \tag{6.10}$$

where y_i^w is the index of the i^{th} correct word.

6.2.3 Training Procedure

So far we have described MUDE which includes a character-level sequence-to-sequence model and a word-level sequential learning model. To train both models simultaneously, we design a loss function for the whole framework as follows:

$$\mathcal{L} = \mathcal{L}_{pred} + \beta \mathcal{L}_{seq2seq} \tag{6.11}$$

where β is a hyperparameter that controls the contribution of the character-level sequence-to-sequence model. Since the major goal of the framework is to predict the correct word given the noised word, we decrease the value of β gradually as the training proceeds to allow the optimizer increasingly focus on improving the word prediction performance.

6.2.3.1 Test Stage

As shown in Figure 6.1, in the test stage, we simply remove the decoder of the sequence-to-sequence model and only keep the encoder in the framework.

6.3 Experiment

In this section, we conduct extensive experiments on the spell correction task to verify the effectiveness of MUDE. Next, we firstly introduce the experimental settings, followed by the analysis of the experimental results.

6.3.1 Experimental Settings

6.3.1.1 Data

We use the publicly available Penn Treebank [86] as the dataset. Following the previous work [102], we firstly experiment on 4 different types of noise: Permutation (PER), Deletion (DEL), Insertion (DEL), and Substitution (SUB), which only operate on the internal characters of words, leaving the first and last characters unchanged. Table 6.1 shows a toy example of a noised sentence. These 4 types of noise can cover most of the realistic cases of misspellings and commonly tested in previous works [12, 98]. For each type of noise, we construct a noised dataset from the original dataset by altering all the words that have more than 3 characters with corresponding noise. We use the same training, validation and testing split in [102], which contains 39,832, 1,700 and 2,416 sentences, respectively.

Table 6.1: Toy examples of noised text

Noise Type	Sentence
Correct	An example of noised text
PER	An epaxmle of nsieod txet
DEL	An examle of nosed tet
INS	An edxample of nmoised texut
SUB	An exsmple of npised test

6.3.1.2 Baselines

To show the effectiveness of MUDE, we compare it with two strong and widely used baselines. The first is Enchant [2] spell checker which is based on dictionaries. The second one is scRNN [102]. It is a recurrent neural network based word recognition model and has achieved previous state-of-the-art results on spell correction tasks. This baseline only considers the sequential dependencies in the word level with a recurrent neural network and ignores that of character level. Note that other baselines including CharCNN [107] have been significantly outperformed by scRNN. Thus, we do not include them in the experiments.

6.3.1.3 Implementation Details

Both scRNN and MUDE are implemented with Pytorch. The number of hidden units of word representations is set to be 650 as suggested by previous work [102]. The learning rate is chosen from {0.1, 0.01, 0.001, 0.0001} and β in Equation 6.11 is chosen from {1, 0.1, 0.001} according to the model performance on the validation datasets. The parameters of MUDE are learned with stochastic gradient decent algorithm and we choose RMSprop [115] to be the optimizer as it did in [102]. To make the comparison fair, scRNN is trained with the same settings as MUDE.

6.3.2 Comparison Results

The comparison results are shown in Table 6.2. There are several observations can be made from the table. The first is that model based methods (scRNN and MUDE) achieve much better performance than dictionary based one (Enchant). This is not surprising as model based methods can effectively utilize the sequential dependencies of words in the sentences.

[2] https://abiword.github.io/enchant/

Moreover, MUDE consistently outperforms scRNN in all cases, which we believe attributes to the effective design of MUDE to capture both character and word level dependencies. More detailed analysis of contribution brought by the character-level dependencies will be shown later in this section. In addition, we observe that the difficulty brought by different types of noise varies significantly. Generally, for model based methods, permutation and insertion noises are relatively easier to deal with comparing to deletion and substitution noises. We argue this is because the former ones do not lose any character information. In other words, the original character information is largely preserved with permutation and insertion. On the contrary, both deletion and substitution can cause information loss, which makes it harder to recognize the original words. This again demonstrate how important the character-level information is. Finally, the results also show that in more difficult situations where deletion or substitution noises present, the advantages of the MUDE become even more obvious. This clearly suggests the effectiveness of the MUDE.

Table 6.2: Performance comparison on different types of noise in terms of accuracy (%). Best results are highlighted with bold numbers.

Method	INT	DEL	INS	SUB
Enchant	72.33	71.23	93.93	79.77
scRNN	98.23	91.55	95.95	87.09
MUDE	**98.81**	**95.86**	**97.16**	**90.52**

Next, we take one step further by removing the constraint that the noise will not affect the first and last characters of each word. More specifically, we define 4 new types of noise that are W-PER, W-DEL, W-INS, and W-SUB, which stand for altering a word by permuting the whole word, deleting, inserting, and substituting characters in any position of the word. Similarly, for each type of new noise, we construct a noised dataset. The results are shown in

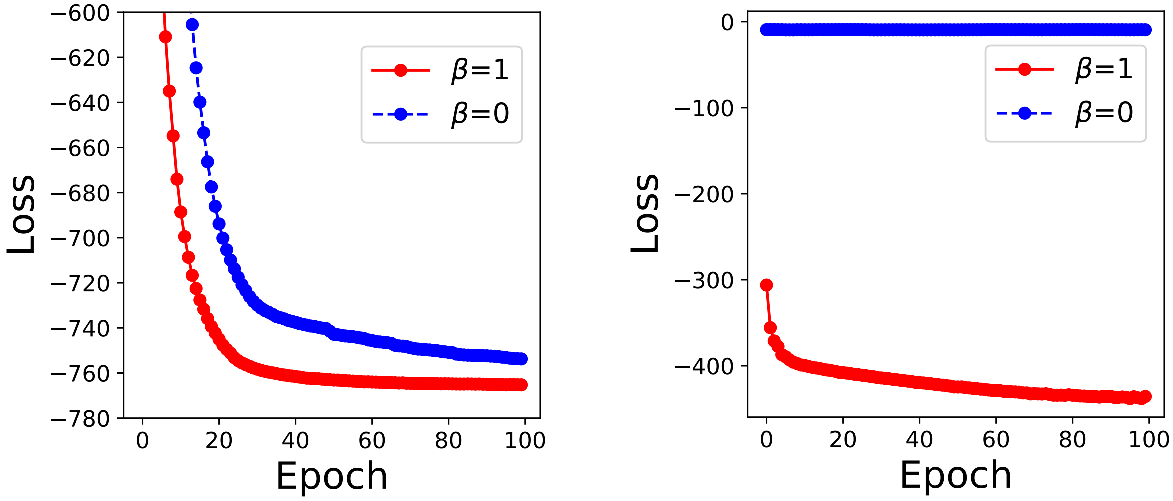


Figure 6.2: Learning curve of MUDE in the training procedure with different β values.

Table 6.3: Performance comparison on different types of noise in terms of accuracy (%). Best results are highlighted with bold numbers.

Method	W-PER	W-DEL	W-INS	W-SUB
Enchant	59.08	69.84	93.77	77.23
scRNN	97.36	89.99	95.96	81.12
MUDE	**98.75**	**93.29**	**97.10**	**85.17**

Table 6.3.

From the table, we observe that firstly, the performance of nearly all methods decreases comparing to that of Table 6.2. This suggests the new types of noise are more difficult to handle, which is expected as they cause more variations of noised words. In fact, without keeping first and last characters of each words, it also becomes a difficult task for human to comprehend the noised sentences [99]. Secondly, MUDE still achieves higher accuracy than other baselines, which is consistent with observations from Table 6.2. More importantly, as the difficulty of the task increases, the advantages of MUDE over scRNN also become more obvious. Take the noise of substitution for example, in Table 6.2, MUDE has around 3.5% absolute accuracy gain over scRNN. When more difficult noise (W-SUB) comes, the performance gain of MUDE becomes 4% as shown in Table 6.3. Such observation is also consistent with previous findings.

In summary, both Table 6.2 and 6.3 clearly demonstrate the robustness of MUDE and its advantages over scRNN which can not utilize the character-level dependencies. Thus, in the next subsection, we conduct analysis on the contribution of character-level dependencies to gain better understanding of MUDE.

6.3.3 Parameter Analysis

In this subsection, we analyze the contribution of character-level dependencies to better word representations by showing the model performance with different β values, which controls the contribution of character-level sequence-to-sequence loss. Specifically, we let the β be 0 and 1. When β is 0, MUDE will totally ignore the character-level dependencies; When β equals to 1, MUDE achieve best accuracy in validation set. The prediction loss and seq2seq loss during the training stage with different β values are shown in Figure 6.2. Note that the trends in Figure 6.2 are similar in all of the cases with the different types noise and we only show that of W-PER case due to the page limitation.

As the upper sub-figure shows, when $\beta = 1$ the prediction loss converges faster and at a lower value comparing to that of case when $\beta = 0$. For the seq2seq loss, it remains constant value when $\beta = 0$ as the model does not learn anything regarding seq2seq task. On the other hand, when $\beta = 1$, the seq2seq loss stably decreases, suggesting that the MUDE is trained to obtain better representation of each word. The obvious positive correlation between these two losses clearly demonstrates the importance of learning character-level dependencies in misspelling correction tasks.

Table 6.4: Generalization analysis results. The best results are highlighted. MEAN shows the average value of each row.

	Test Noise								
Train Noise	PER	W-PER	DEL	W-DEL	INS	W-INS	SUB	W-SUB	MEAN
PER	–	**98.81**	82.55	79.61	92.21	92.37	71.39	69.88	85.70
W-PER	**98.75**	–	81.31	78.3	91.32	91.25	69.55	67.91	84.64
DEL	90.83	90.83	–	**86.02**	79.96	79.97	81.99	76.02	85.18
W-DEL	86.75	86.75	**94.08**	–	78.83	78.87	80.35	79.07	84.74
INS	94.79	94.79	77.3	74.81	–	**97.15**	82.86	80.42	87.41
W-INS	95.67	95.67	78.34	75.95	**97.01**	–	82.96	80.78	**87.91**
SUB	91.71	91.71	88.34	81.49	81.19	81.21	–	**83.65**	86.22
W-SUB	87.05	87.05	83.42	82.42	79.27	79.17	**85.67**	–	83.65

Table 6.5: Data augmentation results. The values that are higher than these of Table 6.4 are bold.

	Test Noise								
	PER	W-PER	DEL	W-DEL	INS	W-INS	SUB	W-SUB	MEAN
W-ALL	96.45	96.45	**94.26**	**93.34**	95.3	95.28	**91.51**	**90.48**	**94.13**

6.3.4 Generalization Analysis

In this subsection, we conduct experiments to understand generalization ability of MUDE. Concretely, we train the framework on one type of noise and test it with a dataset that presents another type of noise. The results are shown in Table 6.4.

From results, we have the following two observations. Firstly, between datasets with similar type of noise, MUDE generalizes quite well (e.g. trained on W-PER and tested on PER), which is not surprising. However, the MUDE trained on one type of noise performs much worse on other types of noise that are very different. These observations suggest that it is hard for MUDE to generalize between noises, which we argue is possibly because of the small overlap between distributions of each type of noise.

Thus, in the next, we apply the commonly used adversarial training method by augmenting all types of noise to train MUDE and test it on each type noise individually. As W-* (*∈ {PER, DEL, INS, SUB}) includes the *, in this experiment, we only combine W-* instead of all types of noise. We denote the new constructed training dataset as W-ALL. The results are shown in Table 6.5. It can be observed from table that the MUDE trained on W-ALL has much better generalization ability (i.e., the mean value is much higher). In addition, it is interesting to see that performance of the MUDE decreases slightly in relatively easy cases where permutation or insertion noise presents while increasing a lot in difficult cases where deletion or substitution noise presents.

6.3.5 Case Study

In this subsection, we take the *Cambridge* sentences which are not the training set as an example to give a qualitative illustration of MUDE's misspelling correction performance. Note that due to the constraint of space, we only show the results of the two types of noise: W-PER and W-INS. The examples are shown in Table 6.6 and 6.7, respectively. We can see from the table that it is quite difficult for even humans to comprehend the noised sentence when first and last characters are also changed. However, MUDE can still recognize almost all of the words. In addition, for both cases, the MUDE has much less errors in the corrected sentence than scRNN, which is consistent with previous quantitative results.

6.4 Related Work

In this section, we briefly review the related literature that is grouped into two categories. The first category includes the exiting works on similar tasks and the second one contains

Table 6.6: An illustrative example of spelling correction outputs for the *Cambridge* sentence noised by W-PER. Words that the models fail to correct are underlined and bold.

Correct	According to a researcher at Cambridge University , it does n't matter in what order the letters in a word are , the only important thing is that the first and last letter be at the right place . The rest can be a total mess and you can still read it without problem . This is because the human mind does not read every letter by itself , but the word as a whole .
Noised	iodrcAngc ot a reeachsr at meigaCdbr srtiinUyve , it seod tn' amrtte in wtah rerdo het tserelt in a rdwo rae , the onyl onmtiaptr ingth si tath hte itfrs dan stla treelt be ta het tgrhi place . hTe rset nca be a aotlt mess dan ouy anc lsilt drae ti tthwuoi lorbmpe . hTsi is aubeecs the huamn dmni edos nto erad evrye lteter by etfisl , but het rdwo sa a eholw .
scRNN	According to a **research** at Cambridge University , it does n't matter in what order the letters in a word are , the only important thing is that the first and last letter be at the right place . The rest can be a total mess and you can still read it without problem . This is because the human mind does not read **very** letter by itself , but the word as a whole .
MUDE	According to a **research** at Cambridge University , it does n't matter in what order the letters in a word are , the only important thing is that the first and last letter be at the right place . The rest can be a total mess and you can still read it without problem . This is because the human mind does not read every letter by itself , but the word as a whole .

previous works that have applied word recognition model to improve the robustness of other NLP systems.

6.4.1 Grammatical Error Correction

Since the CoNLL-2014 shared task [93], Grammatical Error Correction (GEC) has gained great attention from NLP communities [138, 46, 66, 21, 65]. Currently the most effective approaches regard GEC as machine translation problem that translates erroneous sentences to correct sentences. Thus, many methods that are based on statistical or neural machine translation architectures have been proposed. However, most of the existing GEC systems

Table 6.7: An illustrative example of spelling correction outputs for the *Cambridge* sentence noised by W-INS. Words that the models fail to correct are underlined and bold.

Correct	According to a researcher at Cambridge University , it does n't matter in what order the letters in a word are , the only important thing is that the first and last letter be at the right place . The rest can be a total mess and you can still read it without problem . This is because the human mind does not read every letter by itself , but the word as a whole .
Noised	Acxcording to a reysearch at Cazmbridge Univversity , it doesw n't msatter in whmat orderh the letteros in a fword are , the oynly wimportant tghing is tyhat the fircst and ldast legtter be at the rightv placeu . The resty can be a totalp mesus and you can stillb rnead it withougt promblem . Txhis is bebcause the humgan minnd doess not reabd everyb lettfer by itslelf , but the whord as a whvole .
scRNN	according to a **research** at Cambridge University , it does n't matter in what order the letters in a word are , the only important thing is that the first and last **better** be at the right place . The rest can be a total **less** and you can still read it without problem . This is because the human mind does not **rated** every **better** by itself , but the word **a** a whole .
MUDE	According to a **research** at Cambridge University , it does n't matter in what order the letters in a word are , the only important thing is that the first and last letter be at the right place . The rest can be a total **uses** and you can still read it without problem . This is because the human mind does not **bear** every letter by itself , but the word as a whole .

have focused on correction of grammar errors instead of noised spellings. For example, most of words in a wrong sentence in CoNLL-2014 shared task [93] are correct such as 'Nothing is absolute right or wrong', where the only error comes from the specific form 'absolute'. One of the existing works that are most similar to this chapter is scRNN [102], where each word is represented in a fixed 'bag of characters' way. It only consists of a word-level RNN and focused on very easy noise. On the contrary, our proposed framework is more flexible and can obtain meaningful representations that incorporate both character and word-level dependencies. In addition, we have experimented on more difficult types of noise than these in [102] and achieved much better performance.

6.4.2 Denoising Text for Downstream Tasks

Robust NLP systems are becoming increasingly important given the severe consequences adversarial samples can cause [43, 63, 49]. However, previous works have shown that neural machine translation models can be easily broken with words whose characters are permuted [12]. To solve this problem, researchers have found that misspelling correction models can play an extremely effective role [98, 144] in improving the robustness of the systems. For example, Pruthi *et al* [98] firstly applied the pre-trained scRNN model to source sentence to remove noise and then the denoised source sentence was input into the neural translation model to obtain the correctly translated sentence. In addition, Zhou *et al* [144] directly integrated such denoising models into the machine translation system that was trained in an end-to-end approach. In either way, these works suggest that the proposed framework which has demonstrated strong performance can have great potentials in improving the robustness of other NLP systems.

6.5 Discussion

As most of the current NLP systems are very brittle, it is extremely important to develop robust neural models. In this chapter, we have presented a word recognition framework, MUDE, that achieves very strong and robust performance with different types of noise presenting. The proposed framework is able to capture hierarchical structures to obtain effective word representations. Extensive experiments on datasets with various types of noise have demonstrated its superior performance over the exiting popular models.

There are several meaningful future research directions that are worthy exploring. The first is to extend MUDE to deal with sentences where word-level noise presents. For example, in the noised sentences, some of the words might be swapped, dropped, inserted or replaced, etc. In addition, it is also meaningful to improve the generality of MUDE such that it can achieve strong performance with the presence of various types of noise not seen in the training dataset. Another possible future direction is to utilize MUDE to improve the robustness other NLP systems including machine translation, reading comprehension, text classification, etc. Moreover, as this work primarily focuses on English, it would be very meaningful to experiment the proposed framework on other languages. Finally, since dictionary contains very complete information of words, we believe that by incorporating it in our framework, the performance can be further improved.

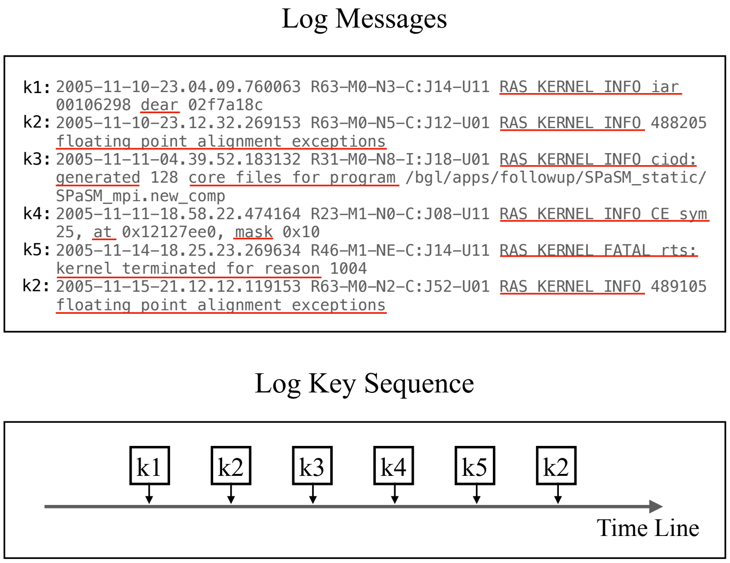
Chapter 7

Modeling Multi-scale Sequential Dependencies

7.1 Introduction

Nowadays, Information and Communication Technology (ICT) has permeated every aspect of our daily life and played a crucial role on the society than ever. While ICT systems have brought unprecedented convenience, when in abnormal states caused by malicious attackers, they could also lead to ramifications including severe loss of economy and social wellbeing [39, 124, 73, 16, 76]. Therefore, it is vital to timely and accurately detect abnormal states of ICT systems such that the loss can be mitigated. Fortunately, with the ubiquitous sensors and networks, ICT systems have generated a large amount of monitoring data [84, 3, 142, 32]. Such data contains rich information and provides us with unprecedented opportunities to understand complex states of ICT systems.

One type of the most important monitoring data is discrete event sequences which can be seen everywhere such as control commands of machine systems, logs of a computer program, transactions of customer purchases and DNA in genetics. Due to the rich information they provide, they have been a valuable source for anomaly detection adopted by both academic research and industry practice [15, 14, 17, 36, 32]. For example, system logs that record the



```
k1: 2005-11-10-23.04.09.760063 R63-M0-N3-C:J14-U11 RAS KERNEL INFO iar
    00106298 dear 02f7a18c
k2: 2005-11-10-23.12.32.269153 R63-M0-N5-C:J12-U01 RAS KERNEL INFO 488205
    floating point alignment exceptions
k3: 2005-11-11-04.39.52.183132 R31-M0-N8-I:J18-U01 RAS KERNEL INFO ciod:
    generated 128 core files for program /bgl/apps/followup/SPaSM_static/
    SPaSM_mpi.new_comp
k4: 2005-11-11-18.58.22.474164 R23-M1-N0-C:J08-U11 RAS KERNEL INFO CE sym
    25, at 0x12127ee0, mask 0x10
k5: 2005-11-14-18.25.23.269634 R46-M1-NE-C:J14-U11 RAS KERNEL FATAL rts:
    kernel terminated for reason 1004
k2: 2005-11-15-21.12.12.119153 R63-M0-N2-C:J52-U01 RAS KERNEL INFO 489105
    floating point alignment exceptions
```

Figure 7.1: An illustrative example of BGL log messages and the corresponding log key sequence. Each log message contains a predefined log key that is underscored by red lines.

detailed messages of run time information by nearly all the modern computer systems are extensively used for anomaly detection [82, 32, 130, 95]. Each log message can be roughly considered as consisting of a predefined constant print statement (also known as “log key” or “message type”) and a specific parameter (also known as “variable”). When the log keys are arranged chronologically according to the recording time, they form a discrete event sequence that can reflect the underlying system state. Figure 7.1 shows an illustrative example of logs from a BlueGene/L supercomputer system (BGL). In this example, there are six log messages that are generated by five corresponding predefined statements (log keys). These log keys form a discrete event sequence. When the system in an abnormal state, the resulted discrete event sequences will deviate from normal patterns. For instance, if “k2” always comes after “k1” in a normal state, then the log key sequence shown in Figure 7.1 may indicate the abnormal state of the system because it shows that “k2”comes directly after “k5”, which is unlikely to happen in the normal state. In this work, we refer to discrete event sequences generated by normal and abnormal system states as normal and abnormal sequences, respectively.

Despite that detecting anomaly for discrete event sequences has attracted much attention [85, 92, 17, 47, 94, 32], it still remains an extremely difficult task due to several intrinsic challenges. The first challenge is from the data imbalance issue that is commonly seen in all kinds of anomaly detection problems. As systems are in normal states in most of the time, abnormal sequences are very rare. This makes the data distributed very unequally in terms of normal and abnormal states. Thus, binary classification models that have achieved great success in the other problems becomes ineffective for anomaly detection. Moreover, in reality we often do not have the prior knowledge about abnormal sequence that further exacerbates the difficulty. The second obstacle comes from the discrete property of events. Unlike continuous sequences where each event are real-valued and have physical meanings, the discrete event sequence consists of discrete symbols, making it hard to capture the relations of events over time. The last but not the least challenge is the sequential nature of the data. In order to determine whether a discrete event sequence is abnormal or not, it is essential to consider each individual event, subsequences of events and the whole sequence simultaneously. This requires dedicated efforts to designing models that not only have strong capability to capture the sequential dependencies but also are flexible to handle sequential dependencies at different scales.

In this chapter, in an attempt to address the aforementioned challenges, we propose an one-class classification framework for event sequence anomaly detection (OC4Seq). It is proposed to directly integrate the anomaly detection objective with a specially designed deep sequence models that explicitly incorporates sequential dependencies at different scales. The main contributions of this work are summarized as follows:

- We identify the importance of multi-scale sequential dependencies in anomaly detection for discrete event sequences empirically.

- We introduce a novel framework (OC4Seq) to explicitly capture multi-scale sequential dependencies and directly optimize the deep sequence models with one-class classification objective.
- We conduct extensive experiments on three datasets to consistently demonstrate that OC4Seq outperforms the state-of-the-art representative methods with a significant margin.

7.2 Problem Statement

Given an event set $\mathcal{E}$ that contains all possible discrete events, an event sequence S^i is defined as $S^i = (e^i_1, e^i_2, \cdots, e^i_{N^i})$, where $e^i_j \in \mathcal{E}$ and N^i is the length of sequence S^i. Each event e^i_j is represented by a categorical value, i.e., $e^i_j \in \mathcal{N}^+$.

With the notations above, the anomaly detection for discrete event sequence problem under the *one-class setting* is formally defined as follows:

Given a set of sequences $\mathcal{S} = \{S^1, S^2, \cdots, S^N\}$, where each sequence S^i is normal, we aim to design a one-class classifier that is able to identify whether a new sequence S is the normal class or not by capturing the underlying multi-scale sequential dependencies in $\mathcal{S}$.

7.3 Preliminaries: One-Class Classifier

In this section, we introduce preliminaries that lay a foundation for our proposed framework. One-Class classifier is a specially designed classifier that is trained with objects of a single class and can predict whether an object belongs to this class or not in the test stage. One of the most widely used one-class classifiers is kernel-based such as One-Class Support Vector

Machines (OC-SVM) [103] and Support Vector Data Description (SVDD) [114]. Both OC-SVM and SVDD are inspired by SVM that tries to maximize the margin between two classes. Next, we use SVDD as an example to illustrate traditional one-class classifiers. SVDD aims at finding a spherically shaped boundary around the given data in the kernel space. Concretely, let the center of the hypersphere be $\mathbf{c}$ and the radius be $R > 0$. The SVDD objective is defined as follows:

$$\min_{R,\mathbf{c},\varepsilon} R^2 + C \sum_{i=1}^{n} \varepsilon_i \tag{7.1}$$
$$s.t. \|\phi(\mathbf{x}_i - \mathbf{c})\|^2 \leq R^2 + \varepsilon_i, \varepsilon_i \geq 0, \quad \forall i$$

where $\mathbf{x}_i$ is the feature vector of i^{th} data and $\varepsilon_i \geq 0$ is a slack variable for $\mathbf{x}_i$ that is introduced to allow the possibility of outliers in the training set and hyperparameter C controls the trade-off between errors ε_i and the volume of the sphere. The objective defined in Equation 7.1 is in primary form and similar to SVM, it is solved in the dual space by using Lagrange multipliers. For more details of the optimization, please refer to the original chapter [114]. Once the R and $\mathbf{c}$ are determined, the points that are outside the sphere will be classified as other classes.

Recently, a deep neural network based one-class classifier called Deep SVDD was introduced in [100]. Inspired by SVDD, it tries to find a minimum hypersphere in the latent space. Unlike SVDD which relies on kernel functions for feature transformation, Deep SVDD takes advantage of deep neural networks to learn data representations. Specifically, the simplified objective that employs a quadratic loss for penalizing the distance of every data point

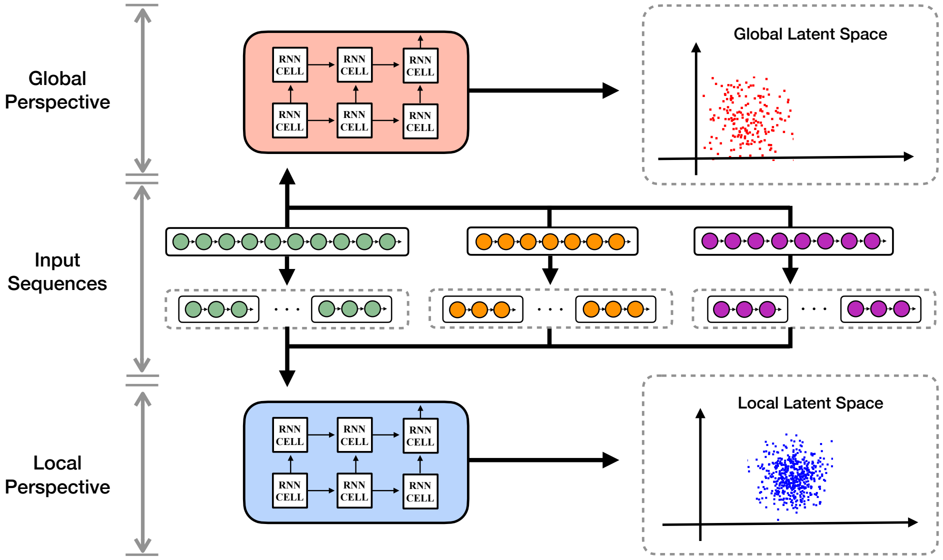


Figure 7.2: The overview of the proposed framework OC4Seq. It consists of two major components that learns the sequential information from global and local perspectives, respectively.

representation to center is defined as:

$$\min_{\mathcal{W}} \frac{1}{n} \sum_{i=1} \|\phi(\mathbf{x}_i) - \mathbf{c}\|^2 + \frac{\lambda}{2} \|\mathcal{W}\|_F^2 \tag{7.2}$$

where $\phi(\cdot)$ is the deep neural network whose parameter are $\mathcal{W}$ and $\|\|_F$ indicates the Frobenius norm. This objective function has been proved with nice theoretical properties [100]. Once the neural networks are trained and center fixed, outliers will be detected similarly as SVDD.

7.4 The Proposed Framework

In this section, we introduce a multi-scale one-class framework for event sequence anomaly detection. An overall of the proposed framework *OC4Seq* is shown in Figure 7.2. It consists of two major components that focus on global and local information in the sequences, respectively. The details of each component are described in the next subsections.

7.4.1 Learning Embeddings for Events

The inputs of the framework are the sequences of events, where each event $\mathbf{e}_t$ is a one-hot vector and $\mathbf{e}(j) = 1, \mathbf{e}(i) = 0 \ \forall i \neq j$ if $\mathbf{e}_t$ is the j^{th} type event of the set $\mathcal{E}$. In real-world scenarios, event space could be very large, i.e., there are tens of thousands of event types. This can lead $\mathbf{e}_t$ to be very high-dimensional and cause notorious learning issues such as sparsity and curse of dimension. In addition, one-hot vector representation makes an implicit assumption that events are independent with each other, which does not hold in most cases. Therefore, we design an embedding layer to embed events into a low-dimension space that can preserve relations between events. To do so, we introduce an embedding matrix $\mathbf{E} \in \mathbb{R}^{d^e \times |\mathcal{E}|}$, where d^e is the dimension of the embedding space and $|\mathcal{E}|$ is the number of event types in $\mathcal{E}$. With this, the representation of $\mathbf{e}_t$ can be obtained as follows:

$$\mathbf{x}_t = \mathbf{E}^T \cdot \mathbf{e}_t \tag{7.3}$$

where $\mathbf{x}_t \in \mathbb{R}^{d^e}$ is the new low-dimensional dense representation vector for $\mathbf{e}_t$. After the embedding layer, the input sequences will be passed into the other components that will be introduced next.

7.4.2 Anomaly Detection from Global Perspective

To detect an anomalous sequence, it is important to learn an effective representation of the whole sequence in the latent space. To this end, we propose to integrate the widely used Gated Recurrent Neural Networks (GRU) [20] with one-class objective function. Specifically, given a normal sequence, i.e., $S^i = (\mathbf{x}_1^i, \mathbf{x}_2^i, \cdots, \mathbf{x}_{N^i}^i)$, the GRU outputs a state vector $\mathbf{h}_{N^i}$ at the final step summarizes all the information in the previous steps, we regard it as the representation of the whole sequence. Please note that the GRU component can be replaced by any sequence learning models such as Long Short-Term Memory (LSTM) [57]. In fact, we empirically found that the two have similar performance. Due to its structural simplicity, we choose GRU over LSTM. More details of component analysis can be found in Section 7.5.

Inspired by the intuition behind the Deep SVDD that all the normal data should be lie within a hypersphere of minimum volume in a latent space, we propose the following objective function to guide the GRU training process:

$$\mathcal{L}_{global} = \min_{\mathbf{\Theta}} \frac{1}{N} \sum_{i=1}^{N} \|\mathbf{h}_{N^i} - \mathbf{c}\|^2 + \lambda \|\mathbf{\Theta}\|_F^2 \tag{7.4}$$

Here $\mathbf{c}$ is a predefined center in the latent space and n is the total number of sequences in the training set. The first term in the objective function employs a quadratic loss for penalizing the distance of every sequence representation to the center $\mathbf{c}$ and the second term is a regularizer controlled by the hyperparameter λ. Therefore, this objective will force the GRU model to map sequences to representation vectors that, on average, have the minimum distances to the center $\mathbf{c}$ in the latent space.

Although GRU is effective to model the whole sequence, it might ignore vital information for event sequence anomaly detection because of the following reason: the abnormal property of a sequence can be caused by only a small abnormal subsequence or even a single abnormal event. However, when the sequence is long, the abnormal information could be overwhelmed by other normal subsequences during the representation learning procedure. This could lead to a very high false negative rate. Therefore, to solve this problem, we design a subsequence learning component that is used to detect the anomaly from the local perspective. In the next subsection, we describe its details.

7.4.3 Anomaly Detection from Local Perspective

In this previous subsection, we introduce to combine GRU and Deep SVDD one-class classification objective to embed the normal sequences in a latent space where they lie within a small distance to a predefined center. However, local information that is vital for anomaly detection could be overwhelmed during this process. Thus, we design the subsequence learning component from the local perspective.

For a given event sequence, we construct subsequences of a fixed size M with a sliding window. Therefore, each subsequence contains its unique local information, which plays an important role to determine whether the whole sequence is abnormal or not. To learn the representation of subsequence, we introduce the local GRU component that will model the sequential dependencies in every subsequence. To be concrete, given a subsequence $\mathbf{x}^i_{t-M+1}, \mathbf{x}^i_{t-M+2}, \cdots, \mathbf{x}^i_t$ of length M, the local GRU processes them sequentially and outputs M hidden states, the last of which is used as the representation of the local subsequence:

$$\mathbf{h}^i_t = \mathrm{GRU}(\mathbf{x}^i_{t-M+1}, \mathbf{x}^i_{t-M+2}, \cdots, \mathbf{x}^i_t) \tag{7.5}$$

Thus, for all subsequences in a sequence, the GRU will obtain a sequence of hidden representations that incorporate sequential dependencies in every local region as follows:

$$\mathbf{h}^i_1, \mathbf{h}^i_2, \cdots, \mathbf{h}^i_{N^i-M} = \mathit{LocalGRU}(\mathbf{x}^i_1, \mathbf{x}^i_2, \cdots, \mathbf{x}^i_{N^i}) \tag{7.6}$$

where LocalGRU is the name for the second GRU model that processes each subsequence. For a normal event sequence, it is intuitive to assume that all of its subsequences are also normal. Thus, we further assume that all the local subsequences should be within a hypersphere in another latent space. To impose this assumption, we design the following objective function to guide the local sequence learning procedure:

$$\mathcal{L}_{local} = \min_{\boldsymbol{\Theta}^L} \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{N^i} \|\mathbf{h}_{N^i_j} - \mathbf{c}^L\|^2 + \lambda \|\boldsymbol{\Theta}^L\|_F^2 \tag{7.7}$$

Here, $\mathbf{c}^L$ is a predefined center of another hypersphere in the latent space and $\boldsymbol{\Theta}^L$ contains all the trainable parameters of LocalGRU. Similarly, the first term penalizes the average distance between all normal subsequences to the center $\mathbf{c}^L$ and the second term is a regularizer.

7.4.4 The Objective Function and Optimization Procedure

In previous subsections, we have introduced components of OC4Seq to detect an abnormal event sequence from both global and local perspectives, respectively. In this subsection, we design an objective function to combine them together. Specifically, given the global and local loss function $\mathcal{L}_{global}$ and $\mathcal{L}_{local}$, the overall objective function of OC4Seq is defined as follows:

$$\min_{\boldsymbol{\Theta}^L, \boldsymbol{\Theta}} \mathcal{L} = \mathcal{L}_{global} + \alpha \mathcal{L}_{local} \tag{7.8}$$

where α is a hyper parameter that controls the contribution from local information in the sequence. This objective enables us to train the framework in an end-to-end manner. The specific optimization procedure is described next.

Optimization. We use stochastic gradient descent (SGD) and its variants (e.g., Adam) to optimize the objective function defined in Equation 7.8. Following previous work [100], to accelerate the training process, the predefined centers $\mathbf{c}$ is computed as follows: given

Table 7.1: The key statistics of *RUBiS*, *HDFS* and *BGL* datasets.

Dataset	# of normal	# of abnormal	# of log keys
RUBiS	11,677	1,000	24
HDFS	558,221	16,838	28
BGL	9,543	985	1,540

the untrained GRU, we firstly feed the sequences in the training set into it and obtain the sequence representation vectors. Then we obtain an average vector by computing the mean value of all representation vectors and use it as $\mathbf{c}$. To obtain $\mathbf{c}^L$, similar process is applied with untrained LocalGRU. Once $\mathbf{c}$ and $\mathbf{c}^L$ are obtained, they remain as constant vectors in the optimization process. The whole training process is done when the objective value converges.

7.5 Experiment

In this section, we conduct extensive experiments to evaluate the proposed framework OC4Seq on one synthetic web logs and two real world system log datasets. Next, we first describe the datasets and experimental settings. Then we present the experimental results and observations. Finally, we conduct qualitative analysis to gain deep understandings on the proposed framework.

7.5.1 Datasets

RUBiS [5]: This dataset is a synthetic web log dataset and was generated by an auction site prototype modeled after eBay.com [5]. Specifically, each log message contains information related to a user web behavior including *user_ id*, *date*, *request_ info*, etc. Following previous works [136, 139], we first parse each log message into a structured representation which consists of a log key and a variable among others. Next, the log keys of a same user are collected following the time order that form an event sequence. Thus, each log key sequence represents a user behavior session in a web server. In addition, we create synthetic anomalous users and their attack cases to obtain abnormal sequences. After the generation process, we are able to collect 11,677 normal sequences and 1,000 abnormal sequences.

Hadoop Distributed File System (HDFS) [130]: This dataset was generated by a Hadoop-based map-reduce cloud environment using benchmark workloads. It contains 11,175,629 log messages, of which 2.9% are labeled as anomalies by Hadoop experts. Each log message involves a block ID, a time stamp and state information. To make the comparison fair, we used the public available dataset[1] processed by [32]. As described in [32], the log messages are firstly parsed into structured text so that a log key is extracted from each log message. In addition, the log keys are sliced into sequences according to the associated block IDs. As a result, there are 558,221 normal sequences and 16,838 abnormal sequences.

BlueGene/L (BGL) [95]: This dataset contains 4,747,936 log messages generated by a BlueGene/L supercomputer system at Lawrence Livermore National Labs (LLNL) in Livermore, California, with 131,072 processors and 32,768GB memory. Each log message contains system information such as type, time stamp, nodes, content, etc. The log messages can be categorized into two types, i.e., non-alert and alert. The non-alert messages are labeled as normal and alert messages are labeled abnormal. Similarly, the log messages are firstly parsed by Drain [54] whose implementation[2] is open sourced by [145]. Following previous work [88], the log keys are sliced using time sliding windows. A sequence is labeled abnormal

[1] https://www.cs.utah.edu/ mind/chapters/deeplog_misc.html

[2] https://github.com/logpai/logparser

Table 7.2: The prediction performance comparison on *RUBiS*, *HDFS* and *BGL* datasets.

Datasets		OC-SVM	PCA	Invariant Mining	DeepLog	OC4Seq
HDFS	F-1 score	0.509	0.634	0.943	0.941	**0.976**
	Precision	0.622	**0.968**	0.893	0.952	0.955
	Recall	9.431	0.471	**1.000**	0.930	0.998
RUBiS	F-1 score	0.351	0.784	0.912	0.935	**0.985**
	Precision	0.220	0.862	0.841	0.885	**0.987**
	Recall	0.869	0.718	**0.996**	0.992	0.983
BGL	F-1 score	0.336	0.423	0.428	0.326	**0.747**
	Precision	0.215	0.269	0.273	0.196	**0.704**
	Recall	0.764	0.993	**1.000**	0.980	0.795

if it contains at least one abnormal message. After processing, there are normal and abnormal sequences.

The statistics of three datasets are summarized in the Table 7.1. Moreover, as stated in Section 7.2, this work focuses on the one-class setting where the training dataset does not contain any abnormal sequence. Therefore each dataset is splitted into training, validation and test sets by the following process. Firstly, we randomly sample the training data from the normal sequence set. Then, we separately split the remaining normal sequences and all the abnormal sequences into validation and test sets with the validation to test ratio 3/7. At last, we combine the two validation/test sets into one.

7.5.2 Baselines

To evaluation the proposed framework, we construct following representative anomaly detection baselines.

- Principle Component Analysis (PCA) [125]. PCA is a classic unsupervised method that has been extensively used for a variety of problems. More recently, it becomes a popular method for anomaly detection [130]. Specifically, it firstly constructs a count matrix $\mathbf{M}$, where each row represents a sequence and each column denotes a log key. Moreover, each entry $\mathbf{M}(i, j)$ indicates the count of j^{th} log key in the i^{th} sequence. Next, PCA learns a transformed coordinate system where the projection lengths of normal sequences are small while these of abnormal sequences are large. Although, it has been shown that PCA can be effective in detecting anomalies especially in reducing false positives [32], it totally ignores the sequential information, which could play an important role for sequence anomaly detection. We use the open sourced implementation[3] [55].

- Invariant Mining (IM) [82]. IM is another popular unsupervised anomaly detection method. It is designed to automatically mine invariants in logs and assumes that discovered invariants can capture the inherent linear characteristics of log flows. Similar to PCA, it firstly constructs a count matrix $\mathbf{M}$. Next, IM learns sparse, integer valued invariants with physical meanings from $\mathbf{M}$. Finally, with the mined invariants, IM makes an invariant hypothesis and sequences that do not satisfy the hypothesis are detected as anomalies. As IM also relies the M, it has similar drawbacks of PCA. The IM used in this work was implemented by [55].

- One-Class SVM (OC-SVM) [103]. OC-SVM is a very effective one-class classier that has been extensively used for anomaly detection [77, 122, 4]. It is especially suited for the our setting that the training set only contains normal data. Specifically, it learns a kernel that maps the normal data into a latent space where all the normal sequence clusters in a small region. Thus, a sequence that does not belong to the cluster is regarded as abnormal. To apply OC-SVM, we firstly need to extract features

[3] https://github.com/logpai/loglizer

from each sequence. In this work, we tried two models to extract features: sequence auto-encoder [25] and bag-of-words [137]. As we empirically found the latter often has better performance, we choose bag-of-words as the feature extractor. The OC-SVM used in this work was implemented with the scikit-learn package[4].

- DeepLog [32]. DeepLog is a state-of-the-art log anomaly detection method. This method is based on a LSTM model which tries to capture the sequential dependencies in sequences. Specifically, by training with normal sequences, it learns to predict the next token given the previously seen tokens in a sequence. During the test stage, for each time step in a sequence, DeepLog will output a probability distribution over all the log keys. If any of the actual tokens is not in the top k candidates, it will regard the sequence as abnormal. Compared to other baselines this method is able to utilize the sequential information and has demonstrated state-of-the-art performance in previous works.

7.5.3 Experimental Settings

Model Selection: For all the methods with hyper-parameters, we use the validation set to select the best value and report the performance on the test set. For DeepLog, we follow the original chapter's suggestion [32]. Specifically, both the h and g are selected from $\{8, 9, 10\}$, which denotes window size and candidates number, respectively. The number of layer is set to be 2 and the number of LSTM hidden units is 64. For OC4Seq, we use the same hyper-parameters as DeepLog and select α that controls the contribution of local subsequence from $\{0.01, 0.1, 1, 10\}$.

[4] https://scikit-learn.org/

Implementation Details: We implemented OC4Seq with Pytorch 1.5 [5]. The model is trained using Adam [69] optimizer with learning rate to be 0.01. The mini-batch size is chosen to be 64 and the model is trained for 100 epochs on a single NVIDIA GEFORCE RTX 2080 card. To encourage reproducible results, we make the code publicly available [6].

Evaluation Metrics: To measure the model performance on anomaly detection, we choose the widely used Precision, Recall and F1 score as the evaluation metrics. They are defined as follows:

$$\begin{aligned} Precision = \frac{TP}{TP+FP} \quad Recall = \frac{TP}{TP+FN} \\ F1 = \frac{2*Precision*Recall}{Precision+Recall} \end{aligned} \tag{7.9}$$

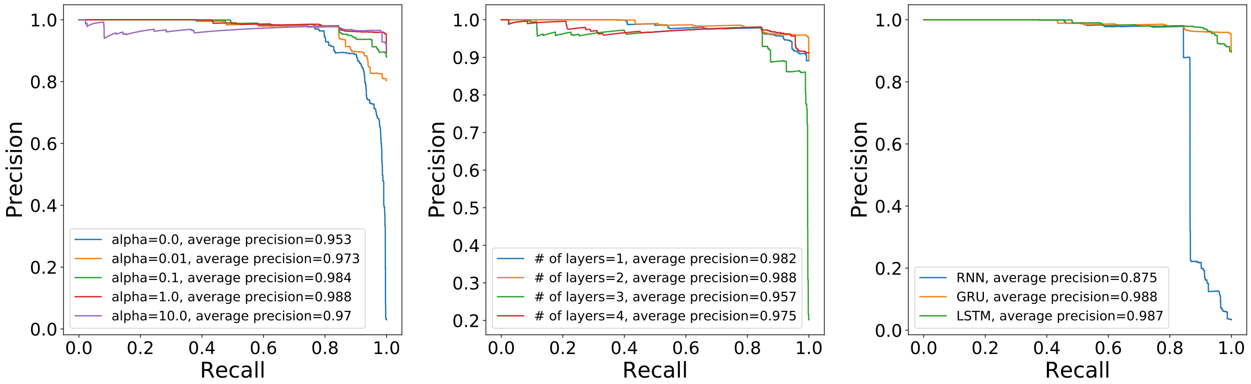
where TP is the abbreviation for True Positives that denotes the number of true abnormal sequences that are detected by the model; FP is the abbreviation for False Positives that measures the number of normal sequences that are regarded as anomalies; FN is False Negative that denotes the number of abnormal sequences the model fails to report. Thus, due to the definition, there is well-known trade-off between precision and recall. On the other hand, the F1 score considers the balance of the two and is often considered as a more comprehensive evaluation metric.

7.5.4 Performance Comparison

The results of all methods on three datasets are shown in Table 7.2. From the table, we made the following observations: 1) On most datasets, OC-SVM has the worse performance. We

[5] https://pytorch.org/

[6] https://github.com/xxxxx



(a) Precision-Recall curves with different alpha.

(b) Precision-Recall curves with # of layers.

(c) Precision-Recall curves with different RNN cell types.

Figure 7.3: Validation Precision-Recall curves on HDFS datasets.

argue that this is because OC-SVM is highly dependent on feature qualities. Unlike dense data where OC-SVM generally performs very well, it is very hard to extract meaningful features from discrete event sequence. 2) IM and DeepLog outperforms PCA significantly in most of the evaluation metrics. This is expected as both IM and DeepLog are designed specifically for anomaly detection for log data. 3) IM and DeepLog generally have comparable results in terms of F-1 score. It is interesting to observe that IM always achieves very impressive Recall while DeepLog is better in Precision. We argue that this difference could be caused by the fact that DeepLog focuses much on the local subsequence information while IM always concentrates on global sequence information. 4) On all the datasets, the proposed framework OC4Seq has achieved best F-1 scores. In addition, when comparing to the second best method, the performance gain brought by OC4Seq is significant. In terms of Precision, OC4Seq still achieved the highest value in most cases. For Recall, OC4Seq was only slightly outperformed by IM which has much lower precision scores. 5) All of the methods performed much worse on BGL datasets than other two datasets. This is because BGL involve much more log keys that can make the task extremely difficult. Moreover, it is interesting to note that the DeepLog method becomes especially ineffective as it heavily relies on the next key prediction which is very difficult when log keys space becomes large. In this challenging case, the improvement from OC4seq over other baselines becomes even more remarkable.

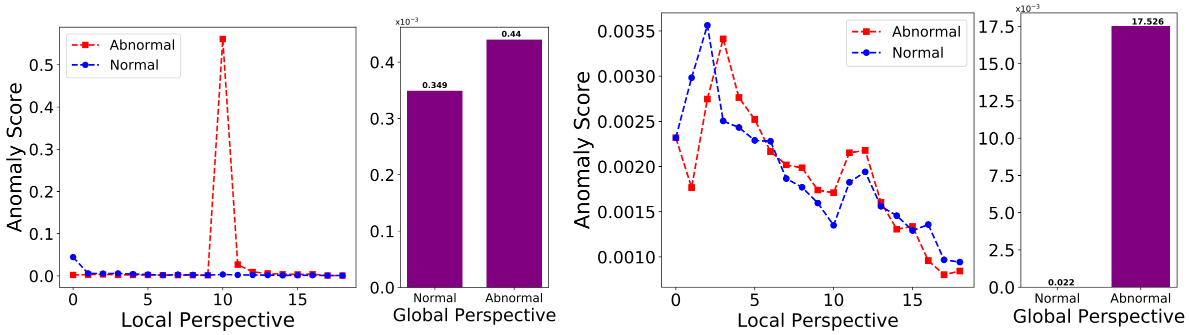
As a summary, from the experimental results on three datasets, our proposed framework OC4Seq demonstrates its superior performance over the baselines. We argue this is because OC4Seq can capture sequential information from both local subsequence and whole sequence and it directly optimizes the anomaly detection objective. Next we design further experiments to gain deeper understandings on OC4Seq.

7.5.5 Parameter Analysis

In this subsection, we analyze key hyper-parameters and components of OC4Seq. We only report the performance on the HDFS validation dataset as we have similar observations on the others. Moreover, the performance is evaluated by Precision-Recall curve as it eliminates the need to choose a specific anomaly threshold and very suitable for datasets with imbalanced label distribution. We also report the area under the Precision-Recall curve (average precision) where the higher the value, the better the performance.

We firstly varies the value of α from $\{0, 0.01, 0.1, 1, 10\}$, which controls the contribution from local subsequence information. The results are shown in Figure 7.3a. As can be seen in the figure, with the increase of α, the performance firstly increases and then decreases. The initial increase demonstrates the importance to incorporate local subsequence information while the latter decrease suggests that the global sequential information is also very essential and should not be overwhelmed by local information.

Next, we varies the number of RNN layers from $\{1, 2, 3, 4\}$ and the results are shown in Figure 7.3b. These results suggest that more layers does not necessary lead to better



(a) Showcase of the importance of Local information.

(b) Showcase of the importance of Global information.

Figure 7.4: The local and global anomaly scores of HDFS log key sequences.

performance as it may cause other issues such as overfitting. Thus, it is important to select a proper value through validation process.

Finally, to investigate how different types of RNN cell affect anomaly detection performance, we experienced on popular cells, i.e., RNN (Vanilla), GRU, LSTM. The results are shown in Figure 7.3c. From the figure, it is easily seen that the LSTM and GRU cells have very similar performance while vanilla RNN achieves much worse results. These results are consistent with previous works [23]. Due to its simpler structure, we choose GRU in OC4Seq.

7.5.6 Case Study

In this subsection, to further understand how the local and global information contribute to anomaly detection, we conduct case studies involving two pairs of representative log key sequences in the HDFS dataset. Specifically, for a sequence, we use the trained model to calculate the anomaly scores of each subsequence and the whole sequence. The higher the anomaly score is, the more likely the sequence is abnormal. The results are shown in Figure 7.4.

In the sub-figure 7.4a, we show the first pair of normal and abnormal sequences. The left panel uses dot lines to demonstrate the anomalous scores for local subsequences. Each dot denotes one anomalous score (y axis) of x^{th} subsequence (x axis). The right panel uses bar to show the anomalous score for the whole sequence (global information). From the figure, we observe that the two sequences have comparable anomalous scores for the whole sequence. Thus it is very difficult to detect the abnormal one purely from the global perspective. However, from the local perspective, we can see that the 10^{th} subsequence of abnormal sequence has a very high anomalous score while the anomalous scores of subsequence of normal sequence are all very low. Therefore, in this case, the local information plays a very important role in detecting anomalies.

In the sub-figure 7.4b, the anomalous scores of the second pair of sequence are shown. Unlike the previous case, the second pair of sequences has very similar anomalous scores for local subsequences. This makes it hard to detect anomaly from the local perspective. However, the abnormal sequence has significantly higher anomalous score from the global perspective than the normal one. Therefore, the global information contributes a lot to detecting the anomaly in this case. From the two cases, we further illustrate the importance of combining both local and global information in a sequence for anomaly detection.

7.5.7 Visualization of Normal and Abnormal Sequences

In this subsection, to gain insights of the one-class classifier objective function, we project the global representations of both normal and abnormal sequences in the HDFS validation set to a two-dimensional space by Local Linear Embedding techniques [31]. The visualization of the two-dimensional space is shown in Figure 7.5. We observe that the normal sequences generally cluster together and lie in a very small region. On the other hand, the abnormal

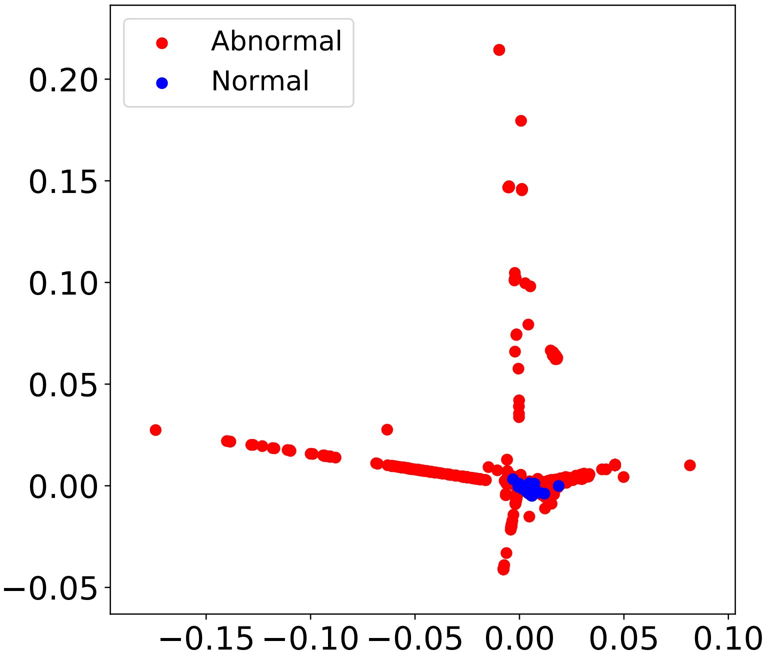


Figure 7.5: The visualization of HDFS datasets. Abnormal and normal data are denoted by red and blue dots, respectively.

sequences spread all over the place. Therefore, the visualization clearly show that by directly minimizing the anomalous score which measures the distance between normal data point and a center point, the one-class classifier objective function is very effective to guide the model to separate the abnormal and normal sequences in the latent space.

7.6 Discussion

In this chapter, we propose the OC4seq, a novel one-class recurrent neural network for discrete event sequence detection. It can deal with multi-scale sequential dependencies and detect anomalies from both local and global perspectives. Specifically, OC4seq incorporates an effective anomaly detection objective that is able to guide the learning process of sequence models. Moreover, it explicitly map the local subsequence and whole sequence into different latent spaces, where the abnormal data points can be easily detected. The proposed OC4seq has consistently shown superior performance than representative baselines in extensive experiments on one synthetic and two benchmark datasets. In addition, through parameter analysis and case studies, the importance of capturing multi-scale sequential dependencies for discrete event sequence anomaly detection has been well demonstrated. Also, the visualization of the sequence representations qualitatively suggests the effectiveness of anomaly detection objectives.

Chapter 8

Conclusions

8.1 Dissertation Summary

In this dissertation, we have described our efforts denoted to sequence learning with side information. Particularly, we have discussed our proposed effective sequences models that capture the most common types of side information and their applications in real-world tasks of various fields.

In Chapter 3, we identified the existence of intrinsic relation of events in certain sequences. To exploit it for sequence learning tasks, we proposed a novel framework SEE that is is able to capture both sequential dependencies and event relations effectively. In addition, our framework has been evaluated extensively with real-world educational and e-commerce sequences. Compared to RNNs that fails to capture the event relation, our proposed framework achieves significantly better performance consistently. Through a carefully designed model component analysis, we showed the contribution of event relation for sequence modeling performance.

In Chapter 4, we focused on capturing the relation between sequences. We firstly refuted the assumption behind most of the existing sequence models that sequences are independently and identically distributed. We showed that in many real-world applications, sequences are inherently related. Given both challenges and opportunities brought by the sequence relation, we devoted our efforts to design a novel sequence model. Our model is based on the Homophily theory that has been verified empirically by many researchers from various fields. Our model can jointly capture the sequential dependencies of events and sequence relation. Through comprehensive experiments, we demonstrated that our model is able to outperform a variety of representative baselines. We also clearly showed that the effectiveness of our model came from its ability to capture sequence relation.

In Chapter 5, we showed that sequences are also associated with temporal information and studied the just-in-time recommendation problem. In addition, we proposed a recommender framework JRec to model the temporal dynamics of customer interest. Compared to traditional recommender systems, JR is not only able to recommend the right products to customers but also do it at the right time. This is attributed to the fact that JRec models both sequential and temporal information of sequences simultaneously. Collaborating with a big e-commerce company, we were able to evaluate our framework with real-world data. The experiment results suggested that JRec outperformed representative baselines in both item recommendation and customer interest prediction tasks.

In Chapter 6, we covered our efforts in designing a sequence model to capture the hierarchical structures in English languages. Specifically, we presented a word recognition model MUDEA to tackle the word recognition problem that plays an extremely important role in robust NLP systems. Our model is able to leverage both character-level and word-level dependencies for this task. We studied our model with extensive experiments. The results showed that MUDE is very robust with different types of word noise presenting. Compared to existing popular word recognition systems, our model has achieved higher accuracy consistently. When the task became more difficult, i.e., with the substitution noise, the performance gain of our model was even more significant. Based on the experiment results, we identified the promise of our model in boosting the robustness of other NLP systems.

In Chapter 7, we discussed the problem of anomaly detection problem for sequential data, which is ubiquitous in information systems. We observed that the state of a sequence depends on sequential dependencies of events of multiple scales. Therefore, we proposed a novel one-class recurrent neural network for this problem. Our model, OC4Seq, is able to capture multi-scale sequential dependencies and detect anomalies from both local and global perspectives. Through extensive experiments on one synthetic and two benchmark anomaly detection datasets, we demonstrated that OC4seq outperformed representative baselines significantly and consistently. To understand our model better, we also conducted a comprehensive analysis of its parameters as well as case studies. The results clearly showed the contribution of multi-scale sequential dependencies to improving both true positive rate and false-negative rate in sequence anomaly detection problem.

In summary, we are excited about the performance boost brought by side information. As most of the research community focuses on learning sequential dependencies, this dissertation provides our unique and pioneering works of exploring and exploiting side information for sequence modeling.

8.2 Future Works

Given the promising results achieved in our works, we believe more dedicated efforts should be devoted in this area. In the following, we outline a few meaningful future directions from two aspects: applications and modeling.

8.2.1 Applications

As stated earlier, sequential data is ubiquitous as well as side information. However, due to the time constraint, we can only apply our models to a limited number of applications. There are many more tasks where proposed models could play a role in boosting state-of-the-art performance. Next, we give two concrete examples.

The first example is the people search problem in social network sites such as Facebook and LinkedIn [60, 61]. The goal of people search is to give the most relevant persons to a search query. Current approaches often consider this problem as a typical ranking problem where user profiles are documents. However, as demonstrated in Chapter 4, the connections between user profiles can also play an important role. Considering the fact that user social connection information is readily available in these social network sites, it is very promising to extend the LinkedRNN proposed in Chapter 4 to improve the current approaches to the people search problem.

The second one is the disease prediction from Electronic Health Record [80], where we aim to predict the disease a patient might develop in the future from his/her records of historical clinical visits. In this case, the patient clinical visits form an event sequence and each visit is one event that has associated a timestamp. The temporal information plays an essential role in this problem because of two reasons. The first comes from the fact that patients usually visit clinics irregularly. This suggests that the historical records are not evenly distributed along the timeline. Second, it is crucial to accurately predict not only the type of disease the patient might have but also when the disease will be developed. Thus, it is a natural idea to extend our work in Chapter 5 to tackle this, which is essentially a sequence modeling with temporal information problem.

8.2.2 Modeling

In this dissertation, we have presented a number of effective models to capture side information for sequence learning. However, there are still many scenarios that require further effort into modeling.

One future direction in this line is to develop unified sequence models to capture various types of side information simultaneously. Our proposed models often focus on one type of side information. However, in many cases, two or more types of side information coexist. For example, the student video-watch sequences in massive open online courses platforms involve both temporal information, i.e., time stamps for each video-watch event and event (lecture) relation [131]. Further, if social connections among students are provided, it would be beneficial to incorporate sequence relation as well. To achieve this, a unified sequence model is necessary. Nevertheless, building unified sequence models is very difficult. The main challenges include keeping a proper balance of information from all perspectives, limiting the model computation complexity, etc. We hope that our proposed models can provide insights and inspirations for future works in this direction.

In addition, our works assume an ideal situation where side information is clean and stable. However, this does not hold in many real-world cases where the side information could contain noise and might change over time. To give a concrete example, let's consider the sequence relation described in Chapter 3, where the sequences are user historical weight records and relations are the social connections among users. It is often the case that user social interest changes frequently. As a result, their social relations are not stable. Thus, it is important to design new models to incorporate both sequence relation and its dynamics. Moreover, there might exist some malicious users who can bring noises to the resulted sequence relations.

Therefore, how to make the model robust to noises is another important problem that need to solve in the future.

The third direction is to improve the interpretability of the models. It is great to see that our proposed models have brought promising results for many sequence learning tasks. Although we have conducted quantitative and qualitative analysis to study where the effectiveness comes from, it is of paramount importance to make the models themselves explainable so that they not only give accurate predictions but also provide the rationale for their behaviors. This can bring huge benefits to both model users and designers. From the user's perspective, knowing the rationale behind the model behaviors is essential for them to trust the model, which is especially vital in health-related domains [118, 35]. From the designer's perspective, interpretability paves a direct way for them to improve model performance further. Existing works in this direction are limited and mainly focus on the sequence part [116, 90]. Thus, dedicated efforts are needed to make our models more interpretable from both sequence and side information perspectives.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Odd Aalen, Ornulf Borgan, and Hakon Gjessing. *Survival and event history analysis: a process point of view.* Springer Science & Business Media, 2008.

[2] Deepak Agarwal, Bee-Chung Chen, and Pradheep Elango. Fast online learning through offline initialization for time-sensitive recommendation. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 703–712. ACM, 2010.

[3] Subutai Ahmad, Alexander Lavin, Scott Purdy, and Zuha Agha. Unsupervised real-time anomaly detection for streaming data. *Neurocomputing*, 262:134–147, 2017.

[4] Mennatallah Amer, Markus Goldstein, and Slim Abdennadher. Enhancing one-class support vector machines for unsupervised anomaly detection. In *Proceedings of the ACM SIGKDD Workshop on Outlier Detection and Description*, pages 8–15, 2013.

[5] Cristiana Amza, Emmanuel Cecchet, Anupam Chanda, Alan L Cox, Sameh Elnikety, Romer Gil, Julie Marguerite, Karthick Rajamani, and Willy Zwaenepoel. Specification and implementation of dynamic web site benchmarks. In *5th Workshop on Workload Characterization*, number CONF, 2002.

[6] Simon Andermatt, Simon Pezold, and Philippe Cattin. Multi-dimensional gated recurrent units for the segmentation of biomedical 3d-data. In *Deep Learning and Data Labeling for Medical Applications*, pages 142–151. Springer, 2016.

[7] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[8] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[9] Dzmitry Bahdanau, Jan Chorowski, Dmitriy Serdyuk, Philemon Brakel, and Yoshua Bengio. End-to-end attention-based large vocabulary speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 4945–4949. IEEE, 2016.

[10] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

[11] Gurkan Bebek. Identifying gene interaction networks. In *Statistical Human Genetics*, pages 483–494. Springer, 2012.

[12] Yonatan Belinkov and Yonatan Bisk. Synthetic and natural noise both break neural machine translation. In *International Conference on Learning Representations*, 2018.

[13] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[14] Suratna Budalakoti, Ashok N Srivastava, Ram Akella, and Eugene Turkov. Anomaly detection in large sets of high-dimensional symbol sequences. 2006.

[15] Suratna Budalakoti, Ashok N Srivastava, and Matthew Eric Otey. Anomaly detection and diagnosis algorithms for discrete symbol sequences with applications to airline safety. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 39(1):101–113, 2008.

[16] Eric Byres and Justin Lowe. The myths and facts behind cyber security risks for industrial control systems. In *Proceedings of the VDE Kongress*, volume 116, pages 213–218, 2004.

[17] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection for discrete sequences: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 24(5):823–839, 2010.

[18] Danqi Chen and Christopher Manning. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 740–750, 2014.

[19] Penghe Chen, Yu Lu, Vincent W Zheng, and Yang Pian. Prerequisite-driven deep knowledge tracing. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 39–48. IEEE, 2018.

[20] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, 2014.

[21] Shamil Chollampatt and Hwee Tou Ng. Connecting the dots: Towards human-level grammatical error correction. In *Proceedings of the 12th Workshop on Innovative Use of NLP for Building Educational Applications*, pages 327–333, 2017.

[22] Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-based models for speech recognition. In *Advances in neural information processing systems*, pages 577–585, 2015.

[23] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[24] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537, 2011.

[25] Andrew M Dai and Quoc V Le. Semi-supervised sequence learning. In *Advances in neural information processing systems*, pages 3079–3087, 2015.

[26] Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 933–941. JMLR. org, 2017.

[27] Matt Davis. Psycholinguistic evidence on scrambled letters in reading, 2012.

[28] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, 2019.

[29] Emily Dinan, Samuel Humeau, Bharath Chintagunta, and Jason Weston. Build it break it fix it for dialogue safety: Robustness from adversarial human attack. *arXiv:1908.06083*, 2019.

[30] Li Dong, Furu Wei, Chuanqi Tan, Duyu Tang, Ming Zhou, and Ke Xu. Adaptive recursive neural network for target-dependent twitter sentiment classification. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 49–54, 2014.

[31] David L Donoho and Carrie Grimes. Hessian eigenmaps: Locally linear embedding techniques for high-dimensional data. *Proceedings of the National Academy of Sciences*, 100(10):5591–5596, 2003.

[32] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1285–1298, 2017.

[33] Nan Du, Hanjun Dai, Rakshit Trivedi, Utkarsh Upadhyay, Manuel Gomez-Rodriguez, and Le Song. Recurrent marked temporal point processes: Embedding event history to vector. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1555–1564. ACM, 2016.

[34] Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. Hotflip: White-box adversarial examples for text classification. *arXiv:1712.06751*, 2017.

[35] Radwa Elshawi, Mouaz H Al-Mallah, and Sherif Sakr. On the interpretability of machine learning-based model for predicting hypertension. *BMC medical informatics and decision making*, 19(1):146, 2019.

[36] German Florez-Larrahondo, Susan M Bridges, and Rayford Vaughn. Efficient modeling of discrete events for anomaly detection using hidden markov models. In *International Conference on Information Security*, pages 506–514. Springer, 2005.

[37] Giorgio Fumera, Ignazio Pillai, and Fabio Roli. Spam filtering based on the analysis of text information embedded into images. *Journal of Machine Learning Research*, 7(Dec):2699–2720, 2006.

[38] Julien Gaillard and Jean-Michel Renders. Time-sensitive collaborative filtering through adaptive matrix completion. In *European Conference on Information Retrieval*, pages 327–332. Springer, 2015.

[39] Robin Gandhi, Anup Sharma, William Mahoney, William Sousan, Qiuming Zhu, and Phillip Laplante. Dimensions of cyber-attacks: Cultural, social, economic, and political. *IEEE Technology and Society Magazine*, 30(1):28–38, 2011.

[40] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1243–1252, 2017.

[41] Eric J Glover, Kostas Tsioutsiouliklis, Steve Lawrence, David M Pennock, and Gary W Flake. Using web structure for classifying and describing web pages. In *Proceedings of the 11th international conference on World Wide Web*, pages 562–569. ACM, 2002.

[42] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE, 2013.

[43] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. Adversarial examples for malware detection. In *European Symposium on Research in Computer Security*, pages 62–79. Springer, 2017.

[44] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864. ACM, 2016.

[45] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864. ACM, 2016.

[46] Roman Grundkiewicz and Marcin Junczys-Dowmunt. Near human-level performance in grammatical error correction with hybrid machine translation. *arXiv:1804.05945*, 2018.

[47] Manish Gupta, Jing Gao, Charu C Aggarwal, and Jiawei Han. Outlier detection for temporal data: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 26(9):2250–2267, 2013.

[48] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.

[49] Han Xu Yao Ma Hao-Chen, Liu Debayan Deb, Hui Liu Ji-Liang Tang Anil, and K Jain. Adversarial attacks and defenses in images, graphs and text: A review. *International Journal of Automation and Computing*, 17(2):151–178, 2020.

[50] Stephen J Hardiman, Nicolas Bercot, and Jean-Philippe Bouchaud. Critical reflexivity in financial markets: a hawkes process analysis. *The European Physical Journal B*, 86(10):442, 2013.

[51] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4):19, 2016.

[52] Alan G Hawkes. Spectra of some self-exciting and mutually exciting point processes. *Biometrika*, 58(1):83–90, 1971.

[53] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[54] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R Lyu. Drain: An online log parsing approach with fixed depth tree. In *2017 IEEE International Conference on Web Services (ICWS)*, pages 33–40. IEEE, 2017.

[55] Shilin He, Jieming Zhu, Pinjia He, and Michael R Lyu. Experience report: System log analysis for anomaly detection. In *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, pages 207–218. IEEE, 2016.

[56] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939*, 2015.

[57] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[58] Cheng-Kang Hsieh, Longqi Yang, Yin Cui, Tsung-Yi Lin, Serge Belongie, and Deborah Estrin. Collaborative metric learning. In *Proceedings of the 26th International Conference on World Wide Web*, pages 193–201. International World Wide Web Conferences Steering Committee, 2017.

[59] Xia Hu, Lei Tang, Jiliang Tang, and Huan Liu. Exploiting social relations for sentiment analysis in microblogging. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 537–546. ACM, 2013.

[60] Jui-Ting Huang, Ashish Sharma, Shuying Sun, Li Xia, David Zhang, Philip Pronin, Janani Padmanabhan, Giuseppe Ottaviano, and Linjun Yang. Embedding-based retrieval in facebook search. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2553–2561, 2020.

[61] Shih-Wen Huang, Daniel Tunkelang, and Karrie Karahalios. The role of network distance in linkedin people search. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 867–870, 2014.

[62] Valerie Isham and Mark Westcott. A self-correcting point process. *Stochastic Processes and Their Applications*, 8(3):335–347, 1979.

[63] Mohit Iyyer, John Wieting, Kevin Gimpel, and Luke Zettlemoyer. Adversarial example generation with syntactically controlled paraphrase networks. *arXiv:1804.06059*, 2018.

[64] Gauri Jain, Manisha Sharma, and Basant Agarwal. Optimizing semantic lstm for spam detection. *International Journal of Information Technology*, 11(2):239–250, 2019.

[65] Jianshu Ji, Qinlong Wang, Kristina Toutanova, Yongen Gong, Steven Truong, and Jianfeng Gao. A nested attention neural hybrid model for grammatical error correction. *arXiv:1707.02026*, 2017.

[66] Marcin Junczys-Dowmunt, Roman Grundkiewicz, Shubha Guha, and Kenneth Heafield. Approaching neural grammatical error correction as a low-resource machine translation task. *arXiv:1804.05940*, 2018.

[67] Komal Kapoor, Karthik Subbian, Jaideep Srivastava, and Paul Schrater. Just in time recommendations: Modeling the dynamics of boredom in activity streams. In

Proceedings of the Eighth ACM International Conference on Web Search and Data Mining, pages 233–242. ACM, 2015.

[68] Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, 2014.

[69] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[70] John Frank Charles Kingman. *Poisson processes*, volume 3. Clarendon Press, 1992.

[71] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[72] Paul Knight, Larry Salmen, and Russell Krajec. Time sensitive inventory sales system, September 23 2004. US Patent App. 10/769,098.

[73] Emanuel Kopp, Lincoln Kaffenberger, and Nigel Jenkinson. *Cyber risk, market failures, and financial stability.* International Monetary Fund, 2017.

[74] Pavel N Krivitsky, Mark S Handcock, Adrian E Raftery, and Peter D Hoff. Representing degree distributions, clustering, and homophily in social networks with latent cluster random effects models. *Social networks*, 31(3):204–213, 2009.

[75] Takeshi Kurashima, Tim Althoff, and Jure Leskovec. Modeling interdependent and periodic real-world action sequences. In *Proceedings of the... International World-Wide Web Conference. International WWW Conference*, volume 2018, page 803. NIH Public Access, 2018.

[76] Monica Lagazio, Nazneen Sherif, and Mike Cushman. A multi-level approach to understanding the impact of cyber crime on the financial sector. *Computers & Security*, 45:58–74, 2014.

[77] Kun-Lun Li, Hou-Kuan Huang, Sheng-Feng Tian, and Wei Xu. Improving one-class svm for anomaly detection. In *Proceedings of the 2003 International Conference on Machine Learning and Cybernetics (IEEE Cat. No. 03EX693)*, volume 5, pages 3077–3081. IEEE, 2003.

[78] Dawen Liang, Jaan Altosaar, Laurent Charlin, and David M Blei. Factorization meets the item embedding: Regularizing matrix factorization with item co-occurrence. In *Proceedings of the 10th ACM conference on recommender systems*, pages 59–66. ACM, 2016.

[79] Zhenjiang Lin, Michael R Lyu, and Irwin King. Pagesim: a novel link-based measure of web page aimilarity. In *Proceedings of the 15th international conference on World Wide Web*, pages 1019–1020. ACM, 2006.

[80] Zachary C Lipton, David C Kale, Charles Elkan, and Randall Wetzel. Learning to diagnose with lstm recurrent neural networks. *arXiv preprint arXiv:1511.03677*, 2015.

[81] Haochen Liu, Tyler Derr, Zitao Liu, and Jiliang Tang. Say what i want: Towards the dark side of neural dialogue models. *arXiv:1909.06044*, 2019.

[82] Jian-Guang Lou, Qiang Fu, Shengqi Yang, Ye Xu, and Jiang Li. Mining invariants from console logs for system problem detection. In *USENIX Annual Technical Conference*, pages 1–14, 2010.

[83] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[84] Pankaj Malhotra, Anusha Ramakrishnan, Gaurangi Anand, Lovekesh Vig, Puneet Agarwal, and Gautam Shroff. Lstm-based encoder-decoder for multi-sensor anomaly detection. *arXiv preprint arXiv:1607.00148*, 2016.

[85] Mirco Marchetti and Dario Stabili. Anomaly detection of can bus messages through analysis of id sequences. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 1577–1583. IEEE, 2017.

[86] Mitchell Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. 1993.

[87] Miller McPherson, Lynn Smith-Lovin, and James M Cook. Birds of a feather: Homophily in social networks. *Annual review of sociology*, 27(1):415–444, 2001.

[88] Weibin Meng, Ying Liu, Yichen Zhu, Shenglin Zhang, Dan Pei, Yuqing Liu, Yihao Chen, Ruizhi Zhang, Shimin Tao, Pei Sun, et al. Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19. International Joint Conferences on Artificial Intelligence Organization*, volume 7, pages 4739–4745, 2019.

[89] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[90] Yao Ming, Panpan Xu, Huamin Qu, and Liu Ren. Interpretable and steerable sequence learning via prototypes. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 903–913, 2019.

[91] Lawrence Mitchell and Michael E Cates. Hawkes process as a model of social interactions: a view on video dynamics. *Journal of Physics A: Mathematical and Theoretical*, 43(4):045101, 2009.

[92] Robert Mitchell and Ray Chen. A survey of intrusion detection in wireless network applications. *Computer Communications*, 42:1–23, 2014.

[93] Hwee Tou Ng, Siew Mei Wu, Ted Briscoe, Christian Hadiwinoto, Raymond Hendy Susanto, and Christopher Bryant. The conll-2014 shared task on grammatical error correction. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*, pages 1–14, 2014.

[94] Rajvardhan Oak, Min Du, David Yan, Harshvardhan Takawale, and Idan Amit. Malware detection on highly imbalanced data through sequence modeling. In *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*, pages 37–48, 2019.

[95] Adam Oliner and Jon Stearley. What supercomputers say: A study of five system logs. In *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, pages 575–584. IEEE, 2007.

[96] Manuel Perea, María Jiménez, Fernanda Talero, and Soraya López-Cañada. Letter-case information and the identification of brand names. *British Journal of Psychology*, 106(1):162–173, 2015.

[97] Chris Piech, Jonathan Bassen, Jonathan Huang, Surya Ganguli, Mehran Sahami, Leonidas J Guibas, and Jascha Sohl-Dickstein. Deep knowledge tracing. In *Advances in Neural Information Processing Systems*, pages 505–513, 2015.

[98] Danish Pruthi, Bhuwan Dhingra, and Zachary C Lipton. Combating adversarial misspellings with robust word recognition. *arXiv:1905.11268*, 2019.

[99] Keith Rayner, Sarah J White, and SP Liversedge. Raeding wrods with jubmled lettres: There is a cost. 2006.

[100] Lukas Ruff, Robert Vandermeulen, Nico Goernitz, Lucas Deecke, Shoaib Ahmed Siddiqui, Alexander Binder, Emmanuel Müller, and Marius Kloft. Deep one-class classification. In *International conference on machine learning*, pages 4393–4402, 2018.

[101] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

[102] Keisuke Sakaguchi, Kevin Duh, Matt Post, and Benjamin Van Durme. Robsut wrod reocginiton via semi-character recurrent neural network. In *AAAI2017*, 2017.

[103] Bernhard Schölkopf, John C Platt, John Shawe-Taylor, Alex J Smola, and Robert C Williamson. Estimating the support of a high-dimensional distribution. *Neural computation*, 13(7):1443–1471, 2001.

[104] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[105] Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. End-to-end memory networks. In *Advances in neural information processing systems*, pages 2440–2448, 2015.

[106] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. Lstm neural networks for language modeling. In *Thirteenth annual conference of the international speech communication association*, 2012.

[107] Ilya Sutskever, James Martens, and Geoffrey E Hinton. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1017–1024, 2011.

[108] Duyu Tang, Bing Qin, and Ting Liu. Document modeling with gated recurrent neural network for sentiment classification. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, pages 1422–1432, 2015.

[109] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1067–1077. International World Wide Web Conferences Steering Committee, 2015.

[110] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1067–1077. International World Wide Web Conferences Steering Committee, 2015.

[111] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. Arnetminer: Extraction and mining of academic social networks. In *KDD'08*, pages 990–998, 2008.

[112] Jiliang Tang, Xia Hu, and Huan Liu. Social recommendation: a review. *Social Network Analysis and Mining*, 3(4):1113–1133, 2013.

[113] Jiliang Tang and Huan Liu. Unsupervised feature selection for linked social media data. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 904–912. ACM, 2012.

[114] David MJ Tax and Robert PW Duin. Support vector data description. *Machine learning*, 54(1):45–66, 2004.

[115] Tijmen Tieleman and Geoffrey Hinton. Rmsprop. *COURSERA: Lecture*, 7017, 2012.

[116] Shikhar Vashishth, Shyam Upadhyay, Gaurav Singh Tomar, and Manaal Faruqui. Attention interpretability across nlp tasks. *arXiv preprint arXiv:1909.11218*, 2019.

[117] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[118] Alfredo Vellido. The importance of interpretability and visualization in machine learning for applications in medicine and health care. *Neural Computing and Applications*, pages 1–15, 2019.

[119] Feng Wang and David MJ Tax. Survey on the attention based rnn model and its applications in computer vision. *arXiv preprint arXiv:1601.06823*, 2016.

[120] Kung-Jeng Wang, YS Lin, and CP Jonas. Optimizing inventory policy for products with time-sensitive deteriorating rates in a multi-echelon supply chain. *International Journal of Production Economics*, 130(1):66–76, 2011.

[121] Xiaolong Wang, Furu Wei, Xiaohua Liu, Ming Zhou, and Ming Zhang. Topic sentiment analysis in twitter: a graph-based hashtag sentiment classification approach. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 1031–1040. ACM, 2011.

[122] Yanxin Wang, Johnny Wong, and Andrew Miner. Anomaly intrusion detection using one class svm. In *Proceedings from the Fifth Annual IEEE SMC Information Assurance Workshop, 2004.*, pages 358–364. IEEE, 2004.

[123] Zhiwei Wang, Tyler Derr, Dawei Yin, and Jiliang Tang. Understanding and predicting weight loss with mobile social networking data. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 1269–1278. ACM, 2017.

[124] Bryan Watkins. The impact of cyber attacks on the private sector. *Briefing Paper, Association for International Affair*, 12, 2014.

[125] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

[126] Marty J Wolf, K Miller, and Frances S Grodzinsky. Why we should have seen that coming: comments on microsoft's tay experiment, and wider implications. *ACM SIGCAS Computers and Society*, 47(3):54–64, 2017.

[127] Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J. Smola, and How Jing. Recurrent recommender networks. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pages 495–503, 2017.

[128] Dongkuan Xu, Wei Cheng, Dongsheng Luo, Yameng Gu, Xiao Liu, Jingchao Ni, Bo Zong, Haifeng Chen, and Xiang Zhang. Adaptive neural network for node classification in dynamic networks. In *ICDM*. IEEE, 2019.

[129] Dongkuan Xu, Wei Cheng, Dongsheng Luo, Xiao Liu, and Xiang Zhang. Spatio-temporal attentive rnn for node classification in temporal attributed graphs. In *IJCAI*, pages 3947–3953, 2019.

[130] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I Jordan. Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 117–132, 2009.

[131] Tsung-Yen Yang, Christopher G Brinton, Carlee Joe-Wong, and Mung Chiang. Behavior-based grade prediction for moocs via time series neural networks. *IEEE Journal of Selected Topics in Signal Processing*, 11(5):716–728, 2017.

[132] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems*, pages 5753–5763, 2019.

[133] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489, 2016.

[134] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489, 2016.

[135] Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V Le. Qanet: Combining local convolution with global self-attention for reading comprehension. *arXiv:1804.09541*, 2018.

[136] Xiao Yu, Pallavi Joshi, Jianwu Xu, Guoliang Jin, Hui Zhang, and Guofei Jiang. Cloudseer: Workflow monitoring of cloud infrastructures via interleaved logs. *ACM SIGARCH Computer Architecture News*, 44(2):489–502, 2016.

[137] Yin Zhang, Rong Jin, and Zhi-Hua Zhou. Understanding bag-of-words model: a statistical framework. *International Journal of Machine Learning and Cybernetics*, 1(1-4):43–52, 2010.

[138] Wei Zhao, Liang Wang, Kewei Shen, Ruoyu Jia, and Jingming Liu. Improving grammatical error correction via pre-training a copy-augmented architecture with unlabeled data. *arXiv:1903.00138*, 2019.

[139] Xu Zhao, Kirk Rodrigues, Yu Luo, Ding Yuan, and Michael Stumm. Non-intrusive performance profiling for entire software stacks based on the flow reconstruction principle. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 603–618, 2016.

[140] Zhengli Zhao, Dheeru Dua, and Sameer Singh. Generating natural adversarial examples. *arXiv:1710.11342*, 2017.

[141] Yin Zheng, Bangsheng Tang, Wenkui Ding, and Hanning Zhou. A neural autoregressive approach to collaborative filtering. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning-Volume 48*, pages 764–773, 2016.

[142] Chong Zhou and Randy C Paffenroth. Anomaly detection with robust deep autoencoders. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 665–674, 2017.

[143] Meizi Zhou, Zhuoye Ding, Jiliang Tang, and Dawei Yin. Micro behaviors: A new perspective in e-commerce recommender systems. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 727–735. ACM, 2018.

[144] Shuyan Zhou, Xiangkai Zeng, Yingqi Zhou, Antonios Anastasopoulos, and Graham Neubig. Improving robustness of neural machine translation with multi-task learning. In *Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1)*, pages 565–571, 2019.

[145] Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, and Michael R Lyu. Tools and benchmarks for automated log parsing. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 121–130. IEEE, 2019.