Knowledge base enrichment via deep neural networks

Bayu Distiawan Trisedya
orcid.org/0000-0002-1672-9483

Submitted in total fulfilment of the requirements of the degree of

Doctor of Philosophy

School of Computing and Information System
THE UNIVERSITY OF MELBOURNE

August 2020

Abstract

A Knowledge base is a large repository that typically stores information about real-world entities. Several efforts have been made to develop knowledge bases in general and specific domains such as DBpedia, YAGO, LinkedGeoData, and Wikidata. These knowledge bases contain millions of facts about entities. However, these knowledge bases are far from complete and mandate continuous enrichment and curation.

In this thesis, we study three common methods to enrich a knowledge base. The first is a *Knowledge Bases Alignment* method that aims to find entities in two knowledge bases that represent the same real-world entity, and then integrates these knowledge bases based on the aligned entities. Many knowledge bases have been created separately for particular purposes with overlapping entity coverage. These knowledge bases are complementary to each other in terms of completeness. We may integrate such knowledge bases to form a more extensive knowledge base for knowledge inferences. The second is a *Relation Extraction* method that aims to extract entities and their relationships from sentences of a corpus and map them to an existing knowledge base. With a large amount of unstructured data sources (i.e., sentences), the relation extraction is an essential method to extract facts from any data source for enriching a knowledge base. The third is a *Description Generation* method that aims to generate a sentence to describe a target entity from its properties in a knowledge base. The generated description can be used to enrich the presentation of the knowledge in a knowledge base, which later can be used in many downstream applications. For example, in question answering, the generated sentence can be used to describe the entity in the answer.

For knowledge bases alignment, we propose an embedding-based entity alignment model. Our model exploits attribute embeddings that capture the similarity between en-

tities in different knowledge bases. We also propose an end-to-end relation extraction model for knowledge base enrichment. The proposed model integrates the extraction and canonicalization tasks. This integration helps the model reduces the error propagation between relation extraction and named entity disambiguation that existing approaches are prone to. For description generation, we propose a content plan based attention model to generate sentences from knowledge base triples in the form of a star-shaped graph. We further propose a graph-based encoder to handle arbitrary-shaped graph for generating entity description. Extensive experiment results show that the proposed methods outperform the state-of-the-art methods in the knowledge base enrichment problems studied.

Declaration

This is to certify that

1. the thesis comprises only my original work towards the Ph.D.,
2. due acknowledgement has been made in the text to all other material used,
3. the thesis is less than 100,000 words in length, exclusive of tables, maps, bibliographies, and appendices.

Bayu Distiawan Trisedya, August 2020

This page intentionally left blank.

Acknowledgements

I am incredibly grateful to my supervisors, Rui Zhang and Jianzhong Qi, for their guidance, knowledge, feedback, and patience throughout my studies. This thesis would have been possible without their support. I would also like to express my special thanks to my advisory committee chair Dr. Sherah Kurnia. Her suggestions have also given me much help in my research projects.

Next, I would like to thank all the co-authors of my papers, especially Prof. Wei Wang and Prof. Gerhard Weikum. Discussions and collaborations with them give me ideas and inspirations toward my research. I am also grateful to have been given the opportunity to visit the Max Planck Institute of Informatics and to conduct research under the supervision of Prof. Weikum.

Furthermore, I would like to thank my colleagues, particularly Yunxiang Zhao, Xiaojie Wang, Yiqing Zhang, Yimeng Dai, Wenkai Jiang, Chuandong Yin, Ang Li, Xinting Huang, Yixin Su, Shiquan Yang, Shuo Zhou, Florin Schimbinschi, Xingjun Ma, Sobia Amjad, Yunzhe Jia, Donia Malekian, Anam Khan, and Sadia Nawaz for the support during my research life in campus.

I would like to acknowledge the financial assistance from the Indonesian Endowment Fund for Education (LPDP), allowing me to explore my research interests, which also contribute to the thesis.

I would like to thank my parents and my sister for their loving support. I also want to thank my family-in-law for being supportive. Lastly, I thank my wife, Yenni, for her endless love and support to me. For my kids, Hasan and Uma, thank you for making me stronger with your laugh and smile.

This page intentionally left blank.

Preface

Portions of this thesis are based on manuscripts:

- Part of the content in Chapter 3 has been published in AAAI 2019: Bayu Distiawan Trisedya, Jianzhong Qi, Rui Zhang (2019). Entity Alignment between Knowledge Graphs Using Attribute Embeddings. In the 2019 Conference on Artificial Intelligence.
- Part of the content in Chapter 4 has been published in ACL 2019: Bayu Distiawan Trisedya, Gerhard Weikum, Jianzhong Qi, Rui Zhang (2019). Neural Relation Extraction for Knowledge Base Enrichment. In the 2019 Annual Meeting of the Association for Computational Linguistics.
- Part of the content in Chapter 5 has been published in AAAI 2020: Bayu Distiawan Trisedya, Jianzhong Qi, Rui Zhang (2020). Sentence Generation for Entity Description with Content-plan Attention. In the 2020 Conference on Artificial Intelligence.
- Part of the content in Chapter 6 has been published in ACL 2018: Bayu Distiawan Trisedya, Jianzhong Qi, Rui Zhang, Wei Wang (2018). GTR-LSTM: A Triple Encoder for Sentence Generation from RDF Data. In the 2018 Annual Meeting of the Association for Computational Linguistics.

I declare that I am the primary author and have contributed $>$ 50% in the papers mentioned above.

This page intentionally left blank.

To my beloved mother.

This page intentionally left blank.

Contents

This page intentionally left blank.

List of Figures

This page intentionally left blank.

List of Tables

This page intentionally left blank.

Chapter 1
Introduction

A knowledge base (KB) is a large repository of facts that are mainly represented as triples. A triple consists of a subject, a predicate, and an object where the predicate indicates the relationship between an entity as the subject and the other entity (or literal) as the object. Here, if the object is an entity, we call the triple a *relationship triple*; if the object is a literal (e.g., geographic coordinate, address, telephone number, etc.), we call the triple an *attribute triple*. A simple fact, "`Melbourne is the capital of Victoria`", can be represented as a graph, as illustrated in Figure 1.1, where the predicate `CapitalOfRegion` connects the entity `Melbourne` as the subject and the entity `Victoria` as the object. The collections of triples in a knowledge base form a graph, i.e., a knowledge graph (KG)[1]. A *Uniform Resource Identifier* (URI) is assigned to each entity as an identifier. Since this form offers simple representations that can be easily interpreted by machines, knowledge bases become essential resources for many applications. For example, Google's Hummingbird algorithm uses a knowledge base to improve its search engine [65]. Meanwhile, IBM created IBM Watson that beats human champions at Jeopardy quiz by exploiting knowledge bases [40].

In recent years, many general and specific domain knowledge bases have been developed. Among them, the most popular general domain knowledge base is DBpedia [3]. DBpedia mainly contains triples extracted from structured data in Wikipedia, such as infobox. Another popular general domain knowledge base is YAGO [57, 139] that includes temporal and spatial information for each real-world entity in it. Besides general domain knowledge bases, there are also efforts in the development of domain-specific

[1] In this thesis, we use the term `knowledge base` and `knowledge graph` interchangeably

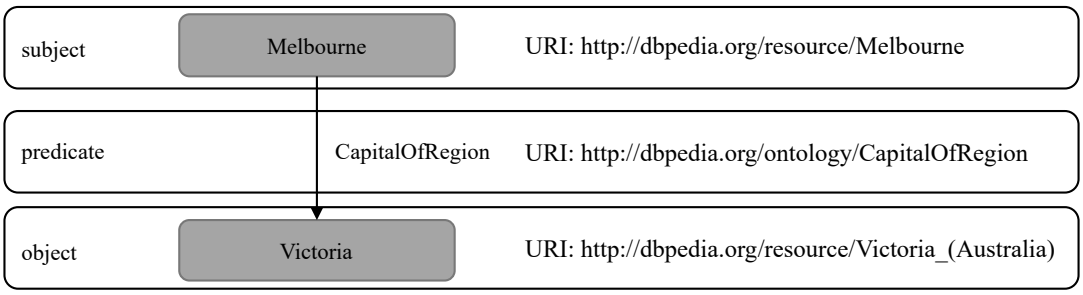


Figure 1.1: Graph representation of a knowledge base triple

knowledge bases such as in geographic domain. Chaves et al. [16] initiated the development of a geographic knowledge base. The follow-up work in the development of a geographic knowledge base is *LinkedGeoData* (LGD) [136]. LGD extracts geographic data from OpenStreetMap (OSM), which is an open-source digital map of the whole world built via crowdsourcing. These knowledge bases contain millions of facts about entities. However, these knowledge bases are far from complete since the information about real-world entities is continuously updated. Hence, knowledge base enrichment methods are essential to achieve a comprehensive knowledge base.

In this thesis, we study three common methods to enrich a knowledge base. The first is a *Knowledge Bases Alignment* method that aims to find entities in two KBs that represent the same real-world entity, and then integrates these KBs based on the aligned entities. Many KBs have been created separately for particular purposes with overlapping entity coverage. These KBs are complementary to each other in terms of completeness. We may integrate such KBs to form a larger KB for knowledge inferences. The second is a *Relation Extraction* method that aims to extract entities and their relationships from sentences in the form of triples and map the elements of the extracted triples to an existing KB. With a large amount of unstructured data sources (i.e., sentences), the relation extraction is an essential method to extract facts from any data source for enriching a KB. We study an end-to-end neural relation extraction technique to extract facts from any website. The third is a *Description Generation* method that aims to generate a sentence to describe a target entity from its *records* (i.e., a set triples that contain the target entity as the subject). The generated descriptions are used to enrich information about entities in a knowledge base, which later can be used in many downstream applications. For example, in ques-

tion answering systems, the generated sentence can be used to describe the entity in the answer. These three common methods for KB enrichment come with different challenges as follows.

In knowledge bases alignment, the main problem to integrate KBs is identifying the entities in different KBs that denote the same real-world entity. We follow current approaches that use graph similarity. In these approaches, a knowledge base is considered as a graph (i.e., a knowledge graph (KG)). The similarity between entities in different knowledge graphs is captured by computing a vector representation (i.e., embeddings) of each entity based on its neighbor in the graph. Existing approaches only consider relationship triples. We observe that many KBs contain large numbers of attribute triples, which have not been explored for entity alignment so far. For example, DBpedia, YAGO, LinkedGeoData, and Geonames contain 47.62%, 62.78%, 94.66%, and 76.78% of attribute triples, respectively. The challenges that need to be handled here are: (i) how to integrate the relationship triples and attribute triples to compute embeddings of an entity; (ii) how to deal with different types of attribute triples such as integer, float, string, etc.

In relation extraction for knowledge base enrichment, there are three main challenges. The first is finding the entity mention, which may appear in multi-word forms. Moreover, the implicit entity mention in a sentence (e.g., pronoun) makes this task even harder. The second is finding the relationship between the extracted entities. There could be multiple extracted entities, and between these entities, there could be multiple relationships, which makes this task non-trivial. The third is mapping the extracted entities and relationships (i.e., triples) to the existing KB. Entity disambiguation is essential to this task since there are entities that share the same name.

Description generation is essentially a reverse problem of the relation extraction task. However, the challenges are different. In description generation, given a target entity that is associated with a set of properties in the form of triples, we aim to generate a sentence to describe the target entity. This task has challenges as follows. The first is selecting salient properties to be mentioned in the sentence. Some properties may not be descriptive, such as `latitude` and `longitude`. The second is arranging the order of properties in the generated sentence. Consider the following two sentences, "`Flinders Street`

`Station is located in Australia, Melbourne"` and `"Flinders Street Station is located in Melbourne, Australia"`. The latter has a more natural order of entity, which is a `City` followed by a `Country`. The third is generating a concise description. We may want to avoid repetition in the generated sentences, e.g., `"Flinders Street Station is located in Melbourne. Flinders Street Station is located in Australia"`.

The three KB enrichment methods above have a high correlation. Recent KB alignment methods are based on the KB embedding model, where these models help the relation extraction methods ensure the validity of the extracted triples. The relation extraction methods and the description generation methods are dual tasks that work in opposite directions. Relation extraction methods aim to extract triples from a text, while description generation methods work in the opposite direction, generating text given a set of triples. The state-of-the-art of these two problems is exploiting end-to-end sequence-to-sequence models such as the encoder-decoder framework. However, these problems possess different challenges.

In this thesis, we overcome the above challenges with novel techniques and make contributions as summarized below.

1.1 Approaches and Contributions

1.1.1 Knowledge Bases Alignment

Earlier studies in knowledge bases alignment use string similarity between properties (i.e., relationship and attribute triples) of entities. For example, RDF-AI [123] implements an alignment framework that consists of pre-processing, matching, fusion, interlink, and post-processing modules, among which the matching module uses fuzzy string matching based on sequence alignment [119], word relation [39], and taxonomic similarity algorithms. However, the string similarity approaches rely on user-defined rules to determine the properties to be compared between the entities. The manually defined rules are error-prone because different entity types may contain a different set of properties. For example, properties such as `latitude` and `longitude` may available for `LOCATION`

type entities but not for PERSON type entities.

Recently, graph similarity (i.e., embedding) approaches are proposed for this task. Such models are built on top of a graph embedding model, such as TransE [11], that learns entity embeddings that capture the similarity between entities in a knowledge graph based on the relationship triples in a KG. To adapt the KG embedding for entity alignment between two KGs, the embedding-based models require both predicate and entity embeddings of two KGs to fall in the same vector space. To address this problem, Chen et al. [19, 20] and Zhu et al. [186] proposed embedding-based alignment models that learn an embedding space for each KG separately and use a transition matrix to map the embedding space from one KG to the other. Their models rely on large numbers of seed alignments (i.e., a seed set of aligned triples from two KGs) to compute the transition matrix. However, the seed alignments between two KGs are rarely available, and hence are difficult to obtain due to expensive human efforts required.

To address the above problems, we propose a fully automatic and effective embedding-based entity alignment embedding model that does not require human intervention either in predicate alignment or in seed entity alignment. Our proposed model includes joint learning of entity, predicate, and attribute embeddings to ensure the resulting embeddings fall in the same vector space. To ensure that the predicate embeddings from two KGs fall into the same vector space, in our model, the predicate embeddings are computed over a predicate proximity graph, which represents a relationship between entity types. Hence, the similarity of predicate embeddings in different KGs can be computed via the relationships between entity types. To yield a unified entity embedding space for two KGs, our model first generates *attribute character embeddings* from the attribute triples and then use this attribute embeddings to shift the entity embeddings of two KGs to the same vector space. We observe that many KGs contain large numbers of attribute triples, which have not been explored for entity alignment so far. The attribute similarity between two KGs helps the attribute embedding to yield a unified embedding space for two KGs. This enables us to use the attribute embeddings to shift the entity embeddings of two KGs into the same vector space and hence allows the entity embeddings to capture the similarity between entities from two KGs.

1.1.2 Relation Extraction for Knowledge Base Enrichment

Relation extraction for knowledge base enrichment includes two subtasks. The first is extracting facts in the form of triples from sentences (i.e., extraction subtask). The second is mapping the extracted triples into the existing knowledge base (i.e., canonicalization subtask). Previous studies handle these subtasks separately. In the extraction subtask, existing methods employ either unsupervised approaches or supervised approaches. Unsupervised approaches [5, 27, 47] use manually defined rules to extract entities and their relationships in a sentence. Here, the extracted entities and relationships are captured in their surface form without canonicalization. Meanwhile, the supervised approaches [78,97,117,176,179] require a pre-processing step to recognize entities in a sentence. Thus, both approaches rely on *Name Entity Disambiguation* (NED) [130] for the canonicalization subtask to map entities and their relationships to the existing KB. This two-stages architecture is prone to error propagation across its two subtasks.

We tackle the problem above by integrating the extraction and the canonicalization subtasks. We propose an end-to-end neural learning model to jointly extract triples from sentences and map them into an existing KB. Our method is based on the encoder-decoder framework [21] by treating the task as a translation of a sentence into a sequence of elements of triples. To capture the multi-word entity names and verbal or noun phrases that denote predicates, we propose a novel form of n-gram based attention. Our attention model computes the n-gram combination of attention weight to capture the verbal or noun phrase context that complements the word level attention of the standard attention model. Thus, our model can better capture the multi-word context of entities and relationships.

1.1.3 Description Generation

In description generation, given a target entity that is associated with a set of properties in the form of triples, we aim to generate a sentence to describe the target entity. This task belongs to the data-to-text generation problem that aims to generate text, e.g., sentences, from structured data, e.g., triples. Here, the triples may form a star-shaped graph with

the target entity (i.e., subject) as the center of the star-shaped graph, the property values (i.e., object in the form of entity or literal) as the points of the star, and the property keys (i.e., relationships) as the edges. Recent studies proposed end-to-end models by adapting the encoder-decoder framework, which is a sequence-to-sequence model used in machine translation. The adaption of the sequence-to-sequence model for data-to-text generation includes representing the input as a sequence. Hence, the order of properties is essential to guide the decoder to generate a good description [156]. However, previous studies [6,81,82] do not explicitly handle the order of input (i.e., properties). In fact, most data sources do not provide sets of properties with a proper order.

We address the issues above by proposing an end-to-end model that jointly learns the entity order in a sentence (i.e., content-planner) and the corresponding sentence as the description of the target entity (i.e., description generator). We integrate the content-plan in the attention model [4] of an encoder-decoder model. The challenge of the integration is to align the learned content-plan and the generated description. To address this problem, we propose the *content-plan-based bag of tokens* attention model by adapting the coverage mechanism [150] to track the order of properties in a content-plan for computing the attention of the attributes. This mechanism helps the attention module of the encoder-decoder model captures the most salient property at each time-step of the description generation phase in a proper order.

We also perform further studies in description generation on arbitrary-shaped graphs as opposed to the star-shaped graphs described above. The star-shaped graph is easy to extract since many KBs allows unnormalized forms, but it is less natural in representing the real-world relationships between entities. For example, the facts expressed by the triples "⟨`John, live in, London`⟩ and ⟨`John, live in, England`⟩" can be represented in a more natural form by the triples "⟨`John, live in, London`⟩ and ⟨`London, capital of, England`⟩". The latter representation may form an arbitrary-shaped graph as opposed to a star-shaped graph, which is more challenging for a machine to process.

To handle arbitrary-shaped graphs, we propose a novel graph-based triple encoder. To capture the relationships both within a triple and between the triples, we propose an

entity traversal scheme based on a topological sort algorithm for the encoding process. Our traversal scheme breaks ties based on entity order in a sentence learned by an entity-order aware translation-based graph embedding model.

1.1.4 Summary of Contributions

For the problem of knowledge bases alignment, we make the following contributions:

- We propose a fully automatic embedding-based entity alignment model to learn the similarity between entities in two KGs with no seed alignments required (neither predicate nor entity seed alignments).
- To compute the entity embeddings, we propose a novel embedding model that integrates entity embeddings with attribute embeddings to learn a unified embedding space for two KGs.
- We propose a novel fully automatic predicate alignment procedure by learning predicate embeddings from a predicate proximity graph of two KGs to capture the similarity between predicates across two KGs automatically.
- We propose a joint learning scheme of entity, predicate, and attribute embeddings to ensure the resulting embeddings fall in the same vector space.
- We evaluate the proposed model over three real KG pairs. The results show that our model outperforms the state-of-the-art models consistently on the entity alignment task by over 40% in terms of *hits*@1.

For the problem of relation extraction for knowledge base enrichment, we make the following contributions:

- We propose an end-to-end model for extracting and canonicalizing triples to enrich a KB. The model reduces error propagation between relation extraction and NED, which existing approaches are prone to.

- We propose an n-gram based attention model to effectively map the multi-word mentions of entities and their relationships into uniquely identified entities and predicates. We propose joint learning of word and entity embeddings to capture the relationship between words and entities for named entity disambiguation. We further propose a modified beam search and a triple classifier to generate high-quality triples.

- We evaluate the proposed model over two real-world datasets. We adapt distant supervision with co-reference resolution and paraphrase detection to obtain high-quality training data. The experimental results show that our model consistently outperforms a strong baseline for neural relation extraction coupled with state-of-the-art NED models.

For the problem of description generation, we make the following contributions:

- We propose an end-to-end model that employs joint learning of content-planning and description generation to handle disordered input for generating a description of an entity from its properties. The model reduces error propagation between the content-planner and the description generator, which two-stage models are prone to. We further propose a content-plan-based bag of tokens attention model to effectively capture salient properties in a proper order based on a content-plan.

- For the arbitrary-shaped graphs, we propose a graph-based triple encoder to optimize the amount of information preserved in the input of the model. The proposed model can handle cycles to capture the relationships both within a triple and between the triples in a KG. To capture the order of entities in a sentence, we present an entity-order aware translation-based graph embedding model.

- We evaluate the proposed models over real-world datasets. The experimental results show that our models consistently outperform state-of-the-art baselines for data-to-text generation.

1.2 Thesis Outline

The rest of the thesis is organized as follows. In Chapter 2, first, we review knowledge base development frameworks in general, then we review research on knowledge bases alignment, relation extraction, and description generation. In Chapter 3, we study the problem of knowledge bases alignment and propose an embedding-based model for aligning entities from two different KBs. In Chapter 4, we study the problem of relation extraction and propose an end-to-end relation extraction for knowledge base enrichment model that integrates the extraction and canonicalization tasks. In Chapter 5, we study the problem of description generation from a set of triples in the form of a star-shaped-graph and propose an order-agnostic data-to-text generation model. In Chapter 6, we study the problem of description generation from arbitrary-shaped graphs and propose a novel graph-based triple encoder. In Chapter 7, we conclude the thesis by summarizing our contributions and discussing future research directions.

Publications out of the Thesis

- Part of the content in Chapter 3 has been published in AAAI 2019: Bayu Distiawan Trisedya, Jianzhong Qi, Rui Zhang (2019). Entity Alignment between Knowledge Graphs Using Attribute Embeddings. In the 2019 AAAI Conference on Artificial Intelligence.

- Part of the content in Chapter 4 has been published in ACL 2019: Bayu Distiawan Trisedya, Gerhard Weikum, Jianzhong Qi, Rui Zhang (2019). Neural Relation Extraction for Knowledge Base Enrichment. In the 2019 Annual Meeting of the Association for Computational Linguistics.

- Part of the content in Chapter 5 has been published in AAAI 2020: Bayu Distiawan Trisedya, Jianzhong Qi, Rui Zhang (2020). Sentence Generation for Entity Description with Content-plan Attention. In the 2020 AAAI Conference on Artificial Intelligence.

- Part of the content in Chapter 6 has been published in ACL 2018: Bayu Distiawan

Trisedya, Jianzhong Qi, Rui Zhang, Wei Wang (2018). GTR-LSTM: A Triple Encoder for Sentence Generation from RDF Data. In the 2018 Annual Meeting of the Association for Computational Linguistics.

This page intentionally left blank.

Chapter 2
Literature Review

THIS chapter reviews existing studies from two aspects. The first is related work on early knowledge base development, which includes algorithms to build a knowledge base from scratch. The second is related work on knowledge base enrichment. We cover three common methods for knowledge base enrichment: (i) knowledge bases alignment, (ii) relation extraction, and (iii) description generation.

2.1 Knowledge Base Development

Building Artificial Intelligence (AI) systems become popular in recent decades. Hence, sharing knowledge between machines becomes an essential part of achieving this purpose. Knowledge bases (KBs) play critical roles in many AI systems since they offer a simple representation for easy data sharing in the form of knowledge graphs (KGs). *Wordnet* [39] is one of the pioneers in knowledge base development. Wordnet contains manually collected lexical knowledge that classifies words and defines the relationship between them, such as *synonym*, *hypernym*, and *meronym*. The development of Wordnet involves experts in linguistic to ensure its quality.

Early general domain knowledge base development approaches are using traditional relation extraction methods, i.e., using rule-based extraction methods. *DBpedia* [3] is one of the large scale knowledge bases that is considered as a central hub of *Linked Open Data*, which is an effort to interlink data on the Internet for knowledge inference through semantic queries. DBpedia mainly contains triples extracted from structured data in Wikipedia, such as infobox. A rule base extractor is used to map infobox tables into

triples in DBpedia. Specific features in a Wikipedia page of an entity such as labels and geographic coordinates are extracted using different rules. Another large-scale knowledge base is *YAGO* [139]. Similar to DBpedia, rule-based methods are used to extract triples from Wikipedia, which becomes the core of YAGO. In the follow-up work, *YAGO2* [57], temporal and spatial data are added to the core of YAGO KB. Temporal data are assigned to entities such as persons, groups, artifacts, and events to denote their existence in time. In contrast, the spatial data are used to denote the location of entities. These data are extracted using regular expressions from Wikipedia, Wordnet, and Geonames[1].

Besides general domain knowledge bases, there are also efforts in the development of domain-specific knowledge bases such as in geographic domain. Chaves et al. [16] initiated the development of a geographic knowledge base. They use standard extraction, transformation, and loading processes (ETL) [132] with predefined rules to extract triples from different sources, including Wikipedia, postal code databases, and gazetteers. In this effort, they cover a limited area, i.e., a single country. The follow-up work in the development of a geographic knowledge base is *LinkedGeoData* (LGD) [136]. LGD extracts geographic data from OpenStreetMap (OSM) and stores them in a relational database. LGD provides a system named *Triplify* to convert database instances into triples.

Another line of work in developing knowledge bases is via crowdsourcing. Crowdsourcing is a method for completing arbitrary tasks by soliciting contributions from a group of human workers [69]. The main problem of this method is determining incentives to persuade users to involve in crowdsourcing. Typically, the incentive for participants is in the form of money. Still, the problem is that there may also be participants who simply want to take the incentives (i.e., money) without making useful contributions. Another incentive scheme is gamification [120]. However, developing an interesting game for gamification may raise another problem since defining the task to be crowdsourced may not be straightforward. Identifying trustworthy participants is another challenge in crowdsourcing. Despite the difficulty of this method, recent studies show successful implementations of crowdsourcing in knowledge base refinement [71,73], where crowdsourcing is used to check the quality of knowledge base data obtained from the tradi-

[1] `http://www.geonames.org/`

tional extraction method.

In summary, there are three common approaches used in the early knowledge base development, including manual approaches [39], traditional extraction methods [3, 57, 139], and crowdsourcing approaches [120]. The manual and crowdsourcing approaches require extensive human labor and are error-prone. Meanwhile, the traditional extraction methods require manually defined rules that work well on a homogeneous data source but may not be applied to different data sources.

2.2 Knowledge Base Enrichment

Existing knowledge bases are far from complete and require continuous enrichment and curation. There are three common approaches to knowledge base enrichment. The first approach is knowledge bases alignment [19, 20], which is done by first aligning entities from the source and target knowledge bases. Then for each aligned entity, all related triples are merged from the source KB to the target KB to have a more comprehensive target KB. The second approach is relation extraction that aims to extract new facts from sentences in the form of KB triples to enrich a knowledge base [78,97,117,176,179]. The third approach is by adding a description for each entity in a knowledge base [6,81,82]. Entity descriptions in a knowledge base are not only used in many downstream applications such as question answering systems and entity linking models but also become valuable resources to provide contextual features for many machine learning models [72, 83, 159]. Thus, description generation techniques for enriching knowledge bases become popular in recent years.

2.2.1 Traditional Methods of Knowledge Base Enrichment

Traditional methods of KB enrichment methods are dominated by rule-based approaches and shallow machine learning models. In KB alignment, earlier approaches, such as LIMES [101], RDF-AI [123], and SILK [158], use string similarity. Other methods, such as LD-Mapper [114] and HolisticEM [110] combine string similarity and graph similarity to compute the alignment. The limitation of these traditional methods is that they rely

on user-defined rules to determine the comparable properties. These manually defined rules are hard to obtained and require significant human efforts.

Traditional approaches to relation extraction can be grouped into three categories: supervised, unsupervised, and bootstrapping. The supervised approach is the most popular technique which achieves relatively high performance. Surdeanu et al. [144] created two-step extraction model. The first step is entity mention detection, and the second step is relation detection between entities extracted from the previous step. Zhou et al. [184] investigated the effect of syntactic and semantic features on relation classifiers using Support Vector Machine (SVM). The unsupervised methods are initiated by Banko et al. [5] that create a self-supervised tuple extractor using the Naïve Bayes classifier. Later, Shinyama and Sekine [131] employed a clustering technique for relation extraction. The other approach is bootstrapping, which use a small amount of example for gaining more data. Ravichandran and Hovy [115] used a small example of question answering data to obtain more extraction patterns. These traditional relation extraction approaches typically use shallow machine learning algorithms, such as Naïve Bayes and SVM. The limitation of these models is they require extensive feature engineering work for the machine learning model, which is labor-intensive and time-consuming.

In traditional description generation, most methods use a pipeline framework that consists of content planning, sentence planning, and surface realization. Bontcheva and Wilks [10] use the pipeline framework to generate sentences from knowledge base triples in the medical domain. Cimiano et al. [23] generate cooking recipes from semantic web data. Duma and Klein [37] learn a sentence template from a parallel triples-text corpus. Similar to traditional relation extraction methods, the traditional description generation models employ hand-crafted rules or a shallow statistical model for content-planning, sentence planning, and surface realization. Thus, these models require significant manual work, which is prone to errors.

2.2.2 Neural Networks for Knowledge Base Enrichment

Neural networks have shown great success in many machine learning tasks such as image recognition, text classification, and video processing. Recent knowledge base en-

richment approaches have adopted neural networks and show better performance than the traditional machine learning models or rule-based approaches. Before we detail three knowledge base enrichment approaches, we discuss neural network models that are commonly used in those approaches.

In this section, we review neural network models that are commonly used in knowledge base enrichment approaches, including word embeddings, knowledge base embeddings, encoder-decoder framework, and graph neural networks.

Word Embeddings

Feature representation is a fundamental part of many machine learning models. Recently, feature representation in the form of a continuous vector space (i.e., embeddings) has attracted the attention of many researchers. For example, in natural language processing and information retrieval, traditional representation such as bag-of-words and n-gram models have been replaced by word embeddings. Such traditional representations have limitations in representing words or sequences of words. Bag-of-words representation ignores the order of words in a sequence and further discards contextual information, while n-gram representation has difficulty in representing long sequences due to the sparsity of the word combinations. In contrast, word embeddings can represent words (or sequences of words) as vectors in a continuous vector space that can preserve the semantic and contextual information of words.

Various word embedding models have been proposed. Bengio et al. [9] introduced word embeddings in their neural language model. Later, pre-trained word embedding models became popular as they can power many downstream tasks. Among them, *Word2vec* [95] and *GloVe* [109] are the early pre-trained embedding models that are widely used by many natural language processing tools. These early pre-trained embedding models are context-independent, i.e., the same vector is used to represent a word across different contexts. For example, the same vector is used to represent the word `apple` in different sentences "`Steve Jobs eats an apple`" and "`Steve Jobs is the founder of Apple`" despite the different meanings the word (i.e., a fruit or a company). To address this problem, context-aware word embedding models are proposed.

Instead of producing a static vector, *ELMo* [111] considers the entire sentence before computing the final embeddings of a word. In contrast, *BERT* [33] computes the word embeddings using a Masked Language Model over the Transformer encoder [153]. These models significantly improve the performance of many downstream applications. Following the success of word embedding techniques, many researchers also study the entity representations of a knowledge base, i.e., knowledge base embeddings.

Knowledge Base Embeddings

Knowledge base embedding models aim to compute the vector representation of entities and relationships in a knowledge base. Translation based models, e.g., *TransE* [11], become one of the pioneers in knowledge base embedding studies. TransE represents a relationship between a pair of entities as a translation between the embeddings of the entities. A triple that consists of $\langle$`subject, predicate, object`$\rangle$, denoted as $\langle s, p, o\rangle$, is represented as $\mathbf{s} + \mathbf{p} \approx \mathbf{o}$. This representation indicates that the embedding of the subject $\mathbf{s}$ is close to the embedding of the object $\mathbf{o}$ by translating $\mathbf{s}$ via the embedding of the predicate $\mathbf{p}$. A scoring function $f(\mathbf{s}, \mathbf{o}) = \|\mathbf{s} + \mathbf{p} - \mathbf{o}\|_2$ is used to measure the plausibility that a triple is incorrect (i.e., *plausibility score*). Here, $\|\mathbf{x}\|_2$ is the L2-norm of a vector x. Wang et al. [162] argued that the triple representation in TransE applies well to 1-to-1 relationships, but have issues on reflexive, N-to-1, 1-to-N, and N-to-N relationships. For example, consider the following reflexive triples $\langle$`John, spouse, Jane`$\rangle$ denoted by $\langle s, p, o\rangle$ and $\langle$`Jane, spouse, John`$\rangle$ denoted by $\langle o, p, s\rangle$. The assumptions in TransE result in $\mathbf{p} = 0$ and $\mathbf{s} = \mathbf{o}$. To address these problems, they proposed *TransH* [162] that normalizes the embeddings of the subject and the object by projecting them into a hyperplane before computing the plausibility score. The TransH scoring function is defined as follows:

$$f(\mathbf{s}, \mathbf{o}) = \|\mathbf{s}_\perp + \mathbf{p} - \mathbf{o}_\perp\|_2 \tag{2.1}$$

$$\mathbf{s}_\perp = \mathbf{s} - \mathbf{w}_\mathbf{r}^\top \mathbf{s} \mathbf{w}_\mathbf{r} \tag{2.2}$$

$$\mathbf{o}_\perp = \mathbf{o} - \mathbf{w}_\mathbf{r}^\top \mathbf{o} \mathbf{w}_\mathbf{r} \tag{2.3}$$

$$\|\mathbf{w_r}\|_2 = 1 \tag{2.4}$$

where $\mathbf{s}_\perp$ and $\mathbf{o}_\perp$ are the projected vector, and $\mathbf{w_r}$ is the hyperplane projection vector.

Lin et al. [79] proposed *TransR* that improves TransH by separating the relationship vector space from the entity vector space. They replace the hyperplane vector $\mathbf{w_r}$ by a projection matrix $\mathbf{M_p}$ that projects the entity embeddings into a corresponding relationship vector space, i.e., $\mathbf{s}_\perp = \mathbf{M_p s}$, and $\mathbf{o}_\perp = \mathbf{M_p o}$. *TransSparse* [60] handles unbalanced relationships in a knowledge base. Some relationships (i.e., predicates) connect many entity pairs while others only connect a few. To address this problem, TransSparse uses a projection matrix that has a sparse degree θ determined by the number of entity pairs connected by the corresponding relationships.

Recent studies also improve TransE by using additional language resources, advanced neural network models, and hierarchy-aware models. For example, *DKRL* [171] and *TEKE* [164] employ additional language resources to compute the entity embeddings. DKRL uses entity descriptions and computes description embeddings using two encoders, including Continuous Bag of Words (CBOW) [95] and Convolution Neural Network encoder [26], respectively. TEKE exploits word co-occurrences that describe entities in a text corpus. In TEKE, the embeddings of the elements of a triple are defined as follows.

$$\mathbf{s} = n\,(s)\,\mathbf{A} + \mathbf{s_b} \tag{2.5}$$

$$\mathbf{o} = n\,(o)\,\mathbf{A} + \mathbf{o_b} \tag{2.6}$$

$$\mathbf{p} = n\,(s,\ o)\,\mathbf{B} + \mathbf{p_b} \tag{2.7}$$

where $\mathbf{A}$ and $\mathbf{B}$ are learned matrices, n(x) and n(x, y) are the point-wise and the pairwise context vector computed based on word co-occurrences, respectively. The subscript $\mathbf{b}$ denotes a bias vector.

Another technique to integrate language resources into KB embeddings is by using a combination of the entity and word embeddings. Malaviya et al. [88] employed a transfer learning approach that first learns word embeddings using BERT, and then concatenates

the word embeddings to the entity embeddings. To compute the structure embeddings, they use *Graph Convolutional Networks* (GCN) [68]. Liu et al. [83] proposed *K-BERT*, which is an extension of BERT trained over triple-enhanced sentences. The input sentence of K-BERT is injected with knowledge base triples to form a knowledge-rich sentence tree. To process the sentence tree, they devise a masked transformer encoder that uses a visible matrix to control the connection between entity mentions in the sentence and the injected triples. He et al. [53] proposed a *knowledge augmented word representation* (KAWR). Their model uses an *entity aware gated recurrent unit* (GERU), which is an extension of the *Gated Recurrent Unit* (GRU) [21] that includes entity attention for computing the hidden state of an input sequence.

Another line of work takes advantage of the flexibility of neural networks to express the relationships between elements of a triple (i.e., the head entity, the predicate, and the tail entity). *ConvE* [32] predicts the embeddings of the tail entity by first concatenating the head entity and predicate embeddings and reshaping them into an input matrix. Then, the matrix is fed into multiple layers of convolutional neural networks connected via residual layers. *ConvKB* [102] and *ConvTransE* [129] follow ConvE. They differ in the stacking and reshaping of the entity and predicate embeddings to provide more interaction among the embeddings. *InteractE* [152] further increases the interaction among embeddings by using feature permutation, multiple reshaping techniques, and circular convolution. *ParamE* [17] and *CoPER* [138] treat the predicate as a parameter (i.e., context) of a projection function that maps the embeddings of the head entity into the embeddings of the tail entities.

Hierarchy-aware models consider multi-hop relationships (e.g., a transitive predicate between two triples $\langle s_1, p_1, o_1 \rangle$ and $\langle o_1, p_2, o_2 \rangle$ indicates a relationship between s_1 and o_2) in a knowledge base for computing the entity and predicate embeddings. *Minerva* [29] uses a reinforcement learning method that gives higher rewards on possible multi-hop walks in a knowledge base. A follow-up work, *Multihop-KG* [77], improves the reward function of Minerva by using a pre-trained embedding model to compute the rewards. *HAKE* [181] transforms the hierarchy in a knowledge base into a polar coordinate system, where the radial coordinate aims to model entities at different levels of the hierarchy

while the angular coordinate aims to distinguish entities at the same level of the hierarchy. *RGHAT* [182] uses two attention mechanisms to model the hierarchy in a knowledge base. The first is predicate-level attention that highlights the most indicative predicate (i.e., property) that represents the subject. The second is entity-level attention that highlights the most indicative entity under the same predicate. Both attention scores are aggregated to predict the object of a triple.

There are also non-translation-based approaches to learn entity embeddings. The Unstructured model [12] does not explicitly represent the relationship embeddings. *RESCAL* [105] and *HolE* [104] use tensor-based factorization and represent relationships with matrices. *NTN* [134] jointly models head and tails entities by combining them using a bilinear tensor operator and maps them to a non-linear hidden layer.

Graph Neural Networks

Many applications produce data in the form of graphs such as knowledge base, social media, e-commerce, etc. The complexity of graph data becomes a challenge in machine learning. Recent studies exploit deep neural networks to solve problems related to unstructured data, including graphs. We review popular Graph Neural Network (GNN) techniques related to knowledge base enrichment models.

The main goal of graph neural networks is to compute node or graph representations (i.e., embeddings) for different tasks such as node classification, edge classification, or graph classification. Node classification aims to predict the label of a node given its embedding learned based on the propagation information from its neighbors. Edge classification is related to the link prediction task. Given the embeddings of two nodes, one may predict the relationship between these nodes. Graph classification aims to predict the label of a graph given a compact vector representation learned from node connections in a graph.

Wu et al. [169] classify graph neural networks into three categories. The first is *recurrent GNN* (RecGNN) that learns node embeddings using a recursive operation, i.e., applying the same set of parameters recursively over nodes in a graph to extract its embeddings. The second is *convolutional GNN* (ConvGNN) that learns the node and edge

embeddings using multiple neural network layers with different weights on each layer. There are two types of ConvGNN: (1) Spectral-based ConvGNN that treats convolution as graph signal processing; (2) Spatial-based ConvGNN that treats convolution as information propagation between nodes in a graph. The third GNN category is Graph Auto Encoder (GAE) that encodes nodes in a graph into vector representations and regenerates the graph based on the encoded representation. Among these categories, ConvGNN is the most popular approach due to its convenience in composing end-to-end models with other neural network models for downstream tasks. Specifically, *Spatial-based ConvGNN* is preferred since it is scalable to a large graph and can handle undirected, weighted, and cyclic graphs.

Graph Convolutional Networks (GCN) [68] is the most popular spatial-based ConvGNN model. As a spatial model, GCN considers convolution as information propagation between nodes in a graph, i.e., a node state is updated by aggregating its own state and its neighbors' state:

$$h_{n_i}^{l} = \sigma\left(\sum_{n_j \in N_{(n_i)} \cup \{n_i\}} \frac{1}{\sqrt{deg(n_i)deg(n_j)}} W^{l} h_{n_j}^{l-1}\right) \tag{2.8}$$

Here, $N_{(n_i)}$ is the neighbors of the entity n_i, $deg(n)$ is the degree of node n, σ is an activation function, W^l is learned parameter of the l-th layer. Multiple layers in GCN are used to capture multi-hop relationships between nodes.

GCN assumes identical structures shared among the entity's neighbors, i.e., each neighbor has the same contribution for computing a node state. However, in real-world graphs, the structures may vary. Follow-up studies address this problem by proposing different weighting mechanisms. *AGCN* (Adaptive Graph Convolutional Neural Networks) [74] uses residual Laplacian matrix and distance metric function to learn the topological structure of a graph. *DGCN* (Dual Graph Convolutional Networks) [187] uses two convolutional layers to compute the local and global consistency, respectively. DGCN also uses positive point-wise mutual information (PPMI) matrix to preserve node co-occurrences captured using random walks as additional features alongside the standard normalized adjacency matrix in GCN. *RGCN* (Relational Graph Convolutional Net-

works) [124] extends GCN by adding relation-specific weight to compute the node states:

$$h_{n_i}^l = \sigma\left(W_i^l h_i^{l-1} + \sum_{r \in R} \sum_{n_j \in N_{(n_i)}} \frac{1}{c_{i,r}} W_r^l h_{n_j}^{l-1}\right) \tag{2.9}$$

where R is the relation (edge) set, $c_{i,r}$ is the normalization constant W_i is the weight matrix for the updated node, and W_r is the neighbor weight based on a specific relation r.

Another popular graph neural network model is *Graph Attention Network* (GAT) [154]. GAT uses an attention model as the weighting mechanism to compute the relative weight of edges connected to a node. The node states in GAT are updated as follows.

$$h_{n_i}^l = \sigma\left(\sum_{n_j \in N_{(n_i)}} \alpha_{ij}^l W_1^l h_{n_j}^{l-1}\right) \tag{2.10}$$

$$\alpha_{ij}^l = softmax\left(g\left(a^\top \left[W_2^l h_{n_i}^{l-1}; W_3^l h_{n_j}^{l-1}\right]\right)\right) \tag{2.11}$$

Here, α_{ij}^l is the attention-based weight that replaces the predefined weight in GCN (i.e., $\frac{1}{\sqrt{deg(n_i)deg(n_j)}}$); $g(.)$ is an activation function such as LeakyReLU; a and w are learned parameters. Similar to the Transformer [153], GAT also uses multi-head attention to compute multiple different weighting schemes that increase the expressiveness of the model in capturing multiple relationships between entities. Follow-up studies propose different techniques to capture multiple relationships between entities. For example, *GAAN* (Gated Attention Network) [180] uses different attentional scores for each attention head. *GeniePath* [84] uses a gating mechanism to control the information propagation in a graph.

Encoder-decoder Framework

The encoder-decoder framework is a sequence-to-sequence learning model popularized by Google in 2014 [21, 146]. This model aims to transform a fixed-length input into a fixed-length output, where the length of the input and output may differ. This model recently achieves significant improvements in many tasks, including text generation that can be used to generate entity description for knowledge base enrichment.

The idea of the encoder-decoder framework is as follows. The encoder is used to encode an input sequence using recurrent neural networks (RNN) [121, 165]. Given a sequence of inputs $\langle x_1, x_2, ..., x_n \rangle$, at each time-step, RNN computes a hidden state h_n^e using the following equation:

$$h_n^e = \sigma(W_1^e x_n + W_2^e h_{n-1}^e) \tag{2.12}$$

where σ is a sigmoid function, and W_1 and W_2 are trained parameters. Intuitively, a hidden state summarizes all information of the previous tokens in a sequence because it recursively includes the hidden state of the previous token h_{n-1}^e to compute h_n^e. Hence, the last hidden state of RNN h_n^e summarizes all information of the whole input. The decoder is used to generate a sequence of outputs $\langle y_1, y_2, ..., y_m \rangle$ using another RNN:

$$h_m^d = \sigma(W_1^d y_{m-1} + W_2^d h_{m-1}^d + W_3^d h_n^e) \tag{2.13}$$

$$y_m = V h_m^d \tag{2.14}$$

where V is the hidden-to-output weight matrix. Here, the summary of input sequence h_n^e is used as a context to generate the output token by token. For example, in machine translation, the encoder first encodes a sentence from a source language into h_n^e, then the decoder uses h_n^e as a reference of the whole representation of the input sentence to generate the corresponding translation word by word in a target language.

One problem with the encoder-decoder framework is that the RNN used in the encoder and the decoder may fail to capture long-range dependencies between words. To handle this problem, Bahdanau et al. [4] proposed an attention mechanism to compute the context vector h_n^e for each time step dynamically. Instead of using the last hidden state of the RNN encoder as a static context, the attention mechanism learns the alignment between words in the source and target sentences, and highlight the most relevant word in the source sentence for generating the output at each time-step. For example, when translating a sentence in English"`I am eating a banana`" to the corresponding sentence in Bahasa Indonesia "`Saya sedang makan pisang`", the model should highlight the word "`eating`" when generating the word "`makan`" since both words have the same

meaning.

At every time-step, the decoder computes the weights of the encoder hidden states $\langle h_1^e, ..., h_n^e \rangle$, which are considered as the representation of each word in the input sequence, and gives the highest score to the most relevant word. These scores are then used to normalized the encoder hidden states. Thus, the most relevant word will have the highest contribution in computing the context vector. The context vector c_m for the decoder at each time-step m is computed as follows.

$$\alpha_i = \frac{\exp({h_m^d}^\top W_1^a h_i^e)}{\sum_{j=1}^{n} \exp({h_m^d}^\top W_2^a h_j^e)} \tag{2.15}$$

$$c_m = \sum_{i=1}^{n} \alpha_i h_i^e \tag{2.16}$$

Attention mechanisms can also be used to solve the out of vocabulary problem. Note that at each time-step of the decoder, the output is a word (or token) that has the highest probability $softmax(y_m)$ in predefined vocabularies. Here, the vocabularies consist of a limited set of words in the target language and may not cover many words, such as the name of a person. To handle this problem, existing text generation models apply a *copy mechanism*. A straightforward copy mechanism is done by copying the most relevant word computed by the attention mechanism (i.e., the word with the highest attention score α) when the decoder generates a `UNK` token (i.e., a unique token that indicates an unknown word generated for a particular time-step). Another copy mechanism [126] uses an additional layer that computes a probability score to decide whether the output should be selected from the vocabularies or be copied from the input token.

Another type of encoder-decoder framework is *Pointer Networks* [157]. Different from the standard encoder-decoder framework that has a fixed size of vocabularies, Pointer Network models deal with the problem of representing variable-length vocabulary, i.e., the candidate outputs are only listed in the given input. Pointer Networks use the softmax probability distributions as pointers to the input for selecting the output at every decoder time-step. Examples of problems that can be solved using pointer networks include convex hull problem, traveling salesman problem, and Delaunay triangulation problem.

The above encoder-decoder frameworks are built upon recurrent neural networks such as LSTM [55] or GRU [21]. The problem with such recurrent models is that the inputs are processed sequentially, thus inhibits parallelization. To address this problem, Vaswani et al. [153] proposed an encoder-decoder model based on a self-attention mechanism named Transformer. They replace the recurrent neural networks in the encoder and decoder with a self-attention model to capture the long-range dependency between words in the source sentence. The difference between the standard attention mechanism and the self-attention mechanism is as follows. In the standard attention mechanism, the previous hidden state of the decoder (e.g., h_m^d in Eq. (2.15)) is used as a query vector over the encoder hidden states as the key/value vectors (i.e., the attention scores of the source tokens are computed based on decoder hidden states). In contrast, in the self-attention mechanism, the query vector is the vector representation of the tokens from the source sentence itself. The Transformer uses multi-head self-attention that expands the capability of a single self-attention to focus on different positions. For example, one self-attention unit may focus on the actual word itself, while the other self-attention unit may focus on a pronoun that refers to it.

2.3 Knowledge Bases Alignment

One of the approaches to enrich knowledge bases is via knowledge bases alignment. Given two knowledge bases, i.e., a *source* KB and a *target* KB, the alignment is done by first identifying the same entity in the source and target KBs then integrating these knowledge bases to form a larger KBs. The integration is done by importing the properties of the same entities from the source KB to the target KB. The main challenge in knowledge bases alignment is to identify the same real-world entities stored in different KBs, i.e., entity alignment. We discuss two groups of commonly used entity alignment approaches. The first is string-similarity-based approaches detailed in Section 2.3.1. The second is embedding-based approaches detailed in Section 2.3.2.

2.3.1 String-Similarity-based Entity Alignment

Earlier entity alignment approaches use string similarity as the main alignment tool. For example, *LIMES* [101] uses the *triangle inequality* to compute similarities between entities from the source and target KBs. First, it creates clusters to group entities on the target KB. Then, the similarities between entities in the source KB and the generated clusters are computed as an approximation. Using these approximations, LIMES avoids to compare every entity pair from the source and target KBs, and hence speed-up the alignment process. Finally, the actual similarity between entities from the source KB and entities in the corresponding cluster on the target KB is computed, and the entity pair with the highest actual string similarity is returned.

RDF-AI [123] implements an alignment framework that consists of pre-processing, matching, fusion, interlink, and post-processing modules. The pre-processing module inspects the ontology consistency and transforms properties into a standard form. The matching module uses fuzzy string matching based on sequence alignment [119], word relation [39], and taxonomic similarity algorithms to compute a similarity score between entities in the source and target KBs. Based on this similarity score, the interlinking module creates a temporary graph that contains entities and their properties from the source KB that correspond to entities in the target KB. The fusion module combines the target KB and the temporary graph from the previous step. Lastly, the post-processor module verifies the resulting ontology consistency.

Another alignment framework, *SILK* [158], allows users to specify the mapping rules using a *Link Specification Language* (LSL). In LSL, users can define the properties and the metrics to be used for entity similarity computation. SILK provides various similarity metrics, including string similarity, numeric similarity, date similarity, and URI equality. To reduce the number of comparisons between entities from two different KBs, SILK provides *rough index pre-matching*. All target resources are indexed based on the values of their properties. This index is used to look up potential matches for a given entity.

There are also studies using graph similarity to improve entity alignment performance. *LD-Mapper* [114] combines string similarity and entity nearest neighbor similarity. *RuleMiner* [106] uses an Expectation-Maximization (EM) algorithm [30] to refine a

set of manually defined entity matching rules. First, a set of matching rules are defined based on the manually collected samples. Then, the EM algorithm iteratively extracts new matching rules and combines them to the existing rules until no further rules extracted. *HolisticEM* [110] constructs a graph of potential entity pairs based on the overlapping attributes and the neighboring entities. From the constructed graph, the local and global properties are propagated using *Personalized Page Rank* to compute the actual similarity of entity pairs.

The string similarity approaches work well when the properties to be compared between the entities are known. However, different knowledge bases may use different property names to store the same property value. Hence, these approaches rely on user-defined rules to determine the comparable properties. The manually defined rules are error-prone because different entity types may contain a different set of properties. For example, properties such as `latitude` and `longitude` may exist in the set of properties of `LOCATION` type entities but not in the set of properties of `PERSON` type entities.

2.3.2 Embedding-based Entity Alignment

Recently, knowledge base embedding models have been proposed to address KB completion tasks [134], which aim to predict missing entities or relationships based on the existing triples in a knowledge base. These models compute a vector representation (i.e., embeddings) of all entities in a KB based on entity nearest neighbors. The embeddings preserve structural information of entities. Entities that share similar neighbors in a KB should have a close vector representation. Hence, the embeddings can be used to compute similarities between entities in different KBs for the alignment. The advancement of KB embedding models motivates researchers to study embedding-based entity alignment. In embedding-based entity alignment, the main challenge is to compute a unified vector space of entity embeddings from two knowledge bases. The existing methods use a seed set of aligned entities (i.e., seed alignments) to force the entity embeddings from two knowledge bases to fall into the same vector space.

Translation Based Alignment Models

Chen et al. [19] proposed *MTransE*, which is an embedding-based model for multilingual entity alignment based on TransE. MTransE uses a *knowledge model* and an *alignment model* to learn the multilingual knowledge base structures. The knowledge model is a standard translation based entity embedding model (i.e., TransE) that computes entity and relationship embeddings of two knowledge bases separately. The alignment model learns a transition matrix to translate both entity and relationships from different embedding spaces (i.e., different KBs) into a unified embedding space over the seed alignments. For computing the transition matrix, they use three strategies: (1) *distance-based axis calibration* that penalizes the alignment based on the distances of cross-lingual counterpart; (2) *translation vector* that encodes cross-lingual transitions into a vector and considers the vector as an additional translation to compute the plausibility score; and (3) *linear transformation* that uses additional neural networks layers to transform the entity and relationship embeddings. In the follow-up work, Chen et al. [20] proposed a generalized affine-map-based model to improve the alignment model of MTransE for handling various forms of invertible transformations, such as translation and scaling.

ITransE [186] uses an iterative method for entity alignment via joint knowledge embeddings. This model consists of three modules. The first is a *knowledge embedding* module that learns entity and relationship embeddings for each knowledge base. The second is a *joint embedding* module that maps the entity and relationship embeddings from different KBs into a joint semantic space according to a seed set of known aligned entities. The third is an *iterative alignment* module that updates the entity and relationship embeddings by taking the high-confident aligned entities found in the previous iteration. Another iterative method is *BootEA* [142] that uses a bootstrapping model. BootEA iteratively assigns alignment of entities from two knowledge bases as pseudo training data to further improve the embeddings and alignment results. The seed alignments and the pseudo training data are combined to train the model for the next iteration. To reduce the error propagation from the pseudo training data, they use a *holistic perspective* model to recompute the probability distribution over the labeled and unlabeled data.

Recently, several techniques are proposed to improve the entity alignment results,

such as exploiting additional information (e.g., the entity types and the description of entities). *JAPE* [141] uses a joint attribute-preserving embedding model for cross-lingual entity alignment. Similar to the previous models, initially, the entity embeddings from two KBs are computed separately. Then, the seed alignments are used to jointly embed the entity embeddings of two KBs into a unified vector space. Their model further refines the alignment by capturing the correlations of attributes via an attribute type similarity model. Chen et al. [18] proposed *KDCoE*, which is a joint learning model between entity embedding and multilingual literal description. For the entity embedding model, they employ TransE. For the description embedding model, they use a self-attention with a gated recurrent unit encoder to learn multilingual word embeddings from the entity descriptions. The description embedding model aims to collocate the embeddings of cross-lingual entities counterparts. They combine the description embeddings and the entity embeddings computed by TransE as a representation of entities and then use them as input for a linear-transformation-based alignment model trained over the seed alignments.

GNN Based Alignment Models

Recent improvements in graph neural networks attract the attention of many researchers to use them in embedding-based entity alignment models such as GCN-Align [163], GMNN [172], MuGNN [14], RDGCN [168], AVR-GCN [175], and AliNet [143]. *GCN-Align* [163] employs standard Graph Convolutional Networks (GCN) [68] to compute the entity embeddings from two different knowledge bases separately. The alignment is computed based on the distances between entities in the embedding space in the final layer of GCN using the seed alignments. To improve the performance of GCN for entity alignment, they combine entity embeddings and attribute type embeddings in the convolutional computation of GCN.

GMNN [172] uses two GCN encoders. The first GCN is similar to GCN-Align, which is a standard GCN to compute entity embeddings of two knowledge bases. The second GCN is combined with an attentive-matching model that first computes the similarity approximation between entities in two knowledge bases, then computes attention scores based on the approximated similarity scores. Later, the attention scores are used

as edge weights in the second GCN. This way, the second GCN may capture the local and global similarities based on the propagated entity embeddings from the first GCN and the attentive-matching models, respectively.

MuGNN [14] highlights two challenges in the GNN based alignment models. The first is the heterogeneity of knowledge base structure problem, i.e., the neighbors of the same real-world entity in two knowledge bases are typically different, and may mislead the embedding learning and the alignment process. The second is the limited seed alignments problem, i.e., the number of the seed alignments are not sufficient to compute high-quality embeddings for alignment. To tackle these problems, they use a two-step method that includes rule-based KB completion and multi-channel Graph Neural Networks entity alignment. The first step, the rule-based KB completion, aims to resolve the structural differences by completing the missing relations using a rule mining system AMIE [43]. The second step is the alignment step that uses a multi-channel Graph Neural Network, which is a combination of GCN and GAT (Graph Attention Network) [154]. In MuGNN, GAT is used to compute a connectivity matrix based on the self entity attention in each KB and the cross-KB attention from the seed alignments, while GCN is used to capture the graph structure based on the connectivity matrix.

Similar to MuGNN, *AliNet* [143] handles the heterogeneity of knowledge base structure problems. Specifically, they consider the multi-hop structure similarity. AliNet consists of two main components, including the Gated Multi-hop Neighborhood aggregation module and the noise reduction module. The aggregator consists of two GCN layers. The first layer is a standard GCN layer to capture the structure of a node's immediate neighbors. The second layer is an attentive GCN layer that uses attention scores to compute the weight of the two-hop neighbors. The noise reduction module is a graph attention network to compute the aggregation of the one-hop and two-hop neighbors.

RDGCN [168] exploits dual relation graphs to improve GCN. The dual relation graph is a graph constructed by combining two KBs and creating additional edges if any relation that shares the same head or tail entities is found. To compute the interaction between the original knowledge base and the dual relation graph, RDGCN uses GAT. The attention scores computed on the dual relation graph are used as weights for the

edges in the original knowledge bases to encode the graphs using GCN. Similar to the existing GNN based alignment models, the final layer of the GCN is used to compute the alignments based on the distances between entities in the seed alignments.

AVR-GCN [175] is an entity alignment framework based on the vectorized relational GCN (VR-GCN), which is the enhanced version of Relational GCN (R-GCN) [124]. Different from the existing GCN-based models that are not computing relation/predicate embeddings, VR-GCN explicitly computes predicate embeddings to be incorporated with the entity embeddings to learn the alignment. Before computing the embeddings using GCN, the entity representation is updated using translation based operations, i.e., if the entity is the head entity, the entity embedding is $\mathbf{t} - \mathbf{r}$, if the entity is the tail entity, the entity embedding is $\mathbf{h} + \mathbf{r}$. To enhance the alignment performance, they use the seed alignments in two ways. The first is they use the seed alignments to compute the objective function similar to the existing GCN-based methods. The second is they create a pseudo graph from the seed alignments then inject them into the original knowledge bases. The expanded graphs are expected to have more shared edges, and hence the embeddings of the same entities from two knowledge bases would be closer to each other.

In summary, the existing embedding-based entity alignment models rely on a set of manually collected seed pairs of aligned entities (i.e., seed alignments). First, the entity and relationship embeddings are computed for each KB separately. Then, a transition matrix is computed using the seed alignments to provide transitions for each embedding to its counterparts. However, the seed alignments between two KBs are rarely available, and hence are difficult to obtain due to expensive human efforts required. To address this problem, in this thesis, we introduce a novel embedding model that first generates attribute embeddings from the attribute triples and then use this attribute embeddings to shift the entity embeddings of two KBs to the same vector space. Our proposed model exploits a large number of attribute triples available in knowledge bases to compute the attribute embeddings, and hence our model does not rely on seed alignments.

2.4 Relation Extraction

Relation extraction plays an important role in extracting structured data from unstructured content, such as text documents. Given a sentence, relation extraction aims to identify entities and their relationship (predicate). Here, the extracted entities and relationships are in the form of knowledge base triples used for knowledge base enrichment. We review two groups of commonly used relation extraction methods. The first is the unsupervised approach that commonly known as open information extraction models detailed in Section 2.4.1. The second is the supervised approach that predicts the relationship of a given entity pair in a sentence. We call this approach entity-aware relation extraction models. This approach is detailed in Section 2.4.2.

2.4.1 Open Information Extraction

Banko et al. [5] introduced the paradigm of *Open Information Extraction* (Open IE) and proposed a pipeline that consists of three stages: learner, extractor, and assessor. The learner uses dependency-parsing information to learn extraction patterns in an unsupervised way. The extractor generates candidate triples by identifying noun phrases as arguments (i.e., entities) and connecting phrases as predicates (i.e., relationships). The assessor assigns a probability score to each candidate triple based on statistical evidence. This approach was prone to extracting incorrect, redundant, and uninformative triples.

Various follow-up studies improve the accuracy of Open IE by adding hand-crafted patterns or by using distant supervision. For example, Fader et al. [38] proposed an Open IE system named *ReVerb* that employs manually defined syntactic constraints expressed in the form of part-of-speech based regular expressions. The syntactic constraints cover 85% relational phrases in English, which help in reducing incoherent and uninformative extraction output. They further employed lexical constraints to avoid over-specified relational phrases. The lexical constraints are based on the intuition that a binary relational phrase should appear in many sentences in a large corpus. Angeli et al. [2] proposed *Stanford Open IE*, which is a classifier for splitting a sentence into smaller parts (i.e., clause) that are semantically entailed the original sentence. The classifier is trained

based on dependency paths. A sentence is traversed along its dependency path, where at each step, the classifier predicts whether the traversed part of the sentence should yield a clause or not. They also specified a small set of manually defined rules to extract a predicate-argument triple.

Mausam et al. [91] proposed a bootstrapping method named *OLLIE* that learns extraction rules based on dependency paths. OLLIE takes a set of high precision seed triples extracted by ReVerb as training data to learn extraction rules. OLLIE also includes a context-analysis step that expands the output representation by adding attribution and clausal modifiers to increase precision. They further presented the follow-up work, including OpenIE4 [90] and OpenIE5.0[2]. *OpenIE4* employs Semantic Role Labeling [22] by taking the extracted verbs as the candidate relationship phrases and the extracted role-labeled arguments as the candidate entities. OpenIE4 also exploits noun-mediated relations from RelNOUN [107] to handle compound nouns for relationship phrases. The latest tool along these lines is *OpenIE5.0* that improves the OpenIE4 by handling the extraction of propositions from conjunctive sentences.

Another work along these lines is the clause based systems. Del Corro et al. proposed *ClausIE* [27], a method that analyzes the clauses in a sentence and derives triples from this structure. First, ClausIE derives coherent clauses from a sentence by using a dependency parser. Then, knowledge base triples are extracted from the derived clauses using basic patterns that are manually defined from English grammar. ClausIE suffers from overly specific extractions that combine unrelated propositions. Gashteovski et al. [47] proposed *MinIE* to advance ClausIE by making the resulting triples more concise, particularly to address the aforementioned problem. MinIE provides four different minimization modes to control the trade-off between precision and recall.

Recently, Stanovsky et al. [137] proposed a supervised learner for Open IE (*supervised Open IE*) by casting relation extraction into sequence tagging. A bidirectional LSTM model is trained to predict the label (entity, predicate, or other) of each token of the input. Another work is *Neural Open IE* [28] that uses an encoder-decoder framework to extract triples. However, this work is not geared to extract the relations of canonicalized enti-

[2] `https://github.com/dair-iitd/OpenIE-standalone`

ties. Another line of study uses neural learning for semantic role labeling [52]. Still, the goal here is to recognize the predicate-argument structure of a single input sentence as opposed to extracting relations from a corpus.

All of these methods generate triples where the subject, the predicate, and the object stay in their surface forms. Therefore, different names and phrases for the same entities result in multiple triples, which would pollute the KB if added this way. The only means to map triples to uniquely identified entities in a KB is by post-processing via entity linking methods [130] or by clustering with subsequent mapping [42].

2.4.2 Entity-aware Relation Extraction

Unlike the Open Information Extraction models that use unsupervised approaches, the training data is often the bottleneck for learning supervised extractors, i.e., lack of training data to train a good supervised extractor. Inspired by the work of Brin et al. [13], state-of-the-art methods circumvent the training bottleneck by employing *distant supervision* [97] that aligns sentences in a text (e.g., Wikipedia article) with seed facts in a KB to collect extensive training data. These alignments are used as training data to build a classifier that takes a pair of entities and a set of aligned sentences as input and predicts the relationship between entities. In this section, we review two common approaches to build such classifiers. The first is the traditional machine learning approach, and the second is the neural networks approach.

Traditional Machine Learning Relation Extraction Models

Traditional relation extraction models [15, 97] learn extraction patterns from seed facts, apply the patterns to extract new fact candidates, iterate this principle, and finally use statistical inference (e.g., a classifier) for assessing the confidence of the triples and reducing the false positive rate. Some of these methods hinge on the assumption that the co-occurrence of a seed fact's entities in the same sentence is an indicator of expressing a semantic relationship between the entities. However, the aligned sentences do not always express a relationship between entities, which can lead to a wrong labeling problem. For

example, the sentence "`Barack Obama visited Honolulu in 2012`" does not indicate the relationship `birth_place` between `Barack Obama` and `Honolulu`.

Follow-up studies [58,117,118,145] overcome the above limitation by various means, including the use of relation-specific lexicons and latent factor models. Riedel et al. [117] introduced an assumption that if an entity pair holds a relation, at least one sentence that contains both entities express the relation (i.e., *at-least-one assumption*). They further proposed a multi-instance learning classifier that computes relevance scores for the aligned sentences and gives the highest score to the most relevant sentence for predicting the relationship. Surdeanu et al. [145] proposed a multi-instance multi-label classifier to accommodate multiple relationships of an entity pair. Suchanek et al. [140] and Sa et al. [122] employed probabilistic-logical inference to eliminate false positives based on constraint solving or Monte Carlo sampling over probabilistic graphical models, respectively.

Neural Relation Extraction Models

The traditional machine learning methods rely on lexical and syntactic features such as part of speech tags, named entity tags, and dependency paths. Recent studies employ neural networks that automatically learn the lexical features for extracting triples from sentences. For example, Nguyen et al. [103] proposed a classifier using *Convolution Neural Networks* (CNN) with multi-sized window kernels. They also employed positional embedding [177] that preserves the distances between words and the entity pair in a sentence. The positional embedding helps the model to handle the long dependency problem in CNN (i.e., the two entities are mentioned far apart in a long sentence). Zeng et al. [176] proposed *PCNN* (Piecewise Convolution Neural Networks) that replaces the standard max-pooling layer by piecewise max-pooling to better capture the lexical structure between entities in a sentence. First, the input sentences are segmented into three parts based on the entity pair position. Then, the piecewise max-pooling returns the maximum value in each segment instead of a single maximum value in the standard max-pooling layer.

Multiple studies further improve PCNN, specifically for addressing the wrong labeling problem. Instead of predicting the relationship between two given entities in a

sentence, current methods use *bag-level* prediction. In bag-level prediction, a set of sentences (i.e., a bag of sentences) that mentions the two given entities is used as input, then the model selects the most representative sentence and considers the other sentences in the bag as noises. Based on the selected sentence, the relationship between the two given entities is predicted.

Lin et al. [78, 80] employed sentence-level attention to address the wrong labeling problem. First, each sentence in the input is encoded into a vector using PCNN. Then, the final representation of the input sentences is computed using the attention-weighted average of all sentences. Jiang et al. [63] combined piecewise max-pooling with cross-sentence max-pooling. The cross-sentence max-pooling relaxes the at-least-once assumption by aggregating the piecewise max-pooling segments from all sentences. Follow-up studies consider further variations: Zhou et al. [185] proposed hierarchical selective attention; Ji et al. [61] employed entity descriptions; Miwa and Bansal [98] incorporated syntactic features; Sorokin and Gurevych [135] proposed context-aware relation extraction by combining the context vector of all relationships in a sentence to predict the relationship between two given entities.

Sentence structures, e.g., dependency trees, provide rich information that helps to improve the performance of relation extraction models. Several neural approaches also exploit this information to improve model performance. Veyseh et al. [155] proposed an end-to-end framework that implicitly learns sentence structures via *ON-LSTM* (Ordered Neuron LSTM) and self-attention unit. ON-LSTM extends the standard LSTM by assigning an attention score for each word in a sentence in the input and forget gates of LSTM. On the other hand, the self-attention unit learns the sentence structures by estimating the long dependency between words in a sentence. Jin et al. [64] exploited the full dependency forest instead of using the 1-best dependency tree generated from an external parser. Their model uses convolution layers to encode the representation of the dependency forest. This representation is combined with the bag representation to predict the relationship between two given entities using feed-forward neural networks.

Li et al. [75] spotted a problem in the bag-level prediction. They found that 80% of bags in the benchmark dataset only contain a single sentence, where roughly 35% of

them are wrongly labeled. This condition may lead the sentence selection mechanisms to learn the wrong patterns. To address this problem, Li et al. [75] proposed a framework that consists of three components. The first component is an entity aware embedding module that dynamically combines position and entity embeddings to highlight entity mentions in a sentence. The second component is a multi-head self-attention mechanism combined with a convolution layer to encode sentences in the bag. The third component is a selective gate module. Instead of using a selective attention mechanism, their model uses a pooling mechanism that assigns lower scores to the wrong labeled sentences and then generates the bag-level representation based on the pooling scores.

Adversarial Networks for Neural Relation Extraction Models

Other works use different techniques to improve relation prediction accuracies, such as adversarial networks and knowledge distillation. Qin et al. [113] employed generative adversarial networks to predict the relationship between two given entities in a sentence. The discriminator is designed to learn two tasks jointly. The first task is differentiating the representation of real data and fake data (i.e., real and fake relation embeddings). The second task is predicting the correct relationship between two given entities. The representation of the real data is computed by a feature encoder that learns the relationship between a pair of entities and their immediate neighbors. Meanwhile, the representation of fake data is generated by multiple layers of fully connected neural networks from a sentence that expresses the relationship between the given entity pair. Zhang et al. [183] employed knowledge distillation via teacher-student network architecture. The teacher networks are used to learn soft-labels from a bipartite graph that maps the entity types of the original entity pairs. The soft-labels is then used to guide the student networks to predict the actual relationship between two given entities.

The aforementioned methods predict the relationships between two given entities. Another line of work jointly predicts a pair of entities and their relationships. Nayak et al. [100] proposed an encoder-decoder relation extraction model with a pointer network-based decoding approach. Instead of generating the output token by token, the decoder generates 5-tuple. The first two elements of the output tuple are the start and end location

of the subject entity in a sentence, the next two elements indicates the location of the object entity, and the last element is the relationships between the subject and the object. Takanobu et al. [148] proposed a hierarchical reinforcement learning method that jointly extracts a pair of entities using a low-level reinforcement learning model and predicts the relationships between the extracted entities using a high-level reinforcement learning model. Xiao et al. [170] proposed a combination of the Transformer model [153] and a reinforcement learning model. The Transformer is used to encode a sentence. They use an LSTM decoder to predict the entity mentions in the sentence. The predicted entities are then used as input for a classifier trained on top of a reinforcement learning model.

The entity-aware models achieve state-of-the-art in relation extraction. However, most of these models are not geared for KB enrichment, as the canonicalization of entities is out of their scope. In practice, named entity disambiguation (NED) is required as a pre-processing step to extract the entities, which may propagate errors for the extraction model that may affect the overall precision and recall. Some methods [122, 140] integrate entity linking (i.e., NED) into their models. However, these models have high computational complexity and rely on modeling constraints and appropriate priors. In this thesis, we address the above problems by proposing a novel end-to-end neural relation extraction model that jointly handles both the extraction and the canonicalization tasks.

2.5 Description Generation

Another approach to enrich a knowledge base is by adding a description for each entity in the knowledge base. Many downstream applications such as question answering and entity linking benefit from these entity descriptions. Recently, many researchers study entity description generation to generate entity descriptions in a knowledge base automatically. Entity description generation aims to produce sentences that describe a target entity from a set of triples of the target entity. This task falls in the area of *text generation*, specifically *data-to-text-generation*, where the data is in the form of triples. In this section, we review two common text generation approaches. The first includes traditional

approaches that use hand-crafted rules and shallow statistical machine learning models (Section 2.5.1). The second includes neural text generation models (Section 2.5.2).

2.5.1 Traditional Text Generation

Traditional text generation [116] models consist of three steps: (1) *content planning*, which is a step to select the data to be expressed, (2) *sentence planning*, which is a step to decide the structures of the sentences to be generated, and (3) *surface realization*, which is a step to generate the final output based on the sentence planning. Earlier studies on content-planning employ hand-crafted rules [36] or a machine learning model as a content classifier [7, 35]. For sentence planning and surface realization, earlier studies proposed template-based models [92,151], machine learning models using various linguistic features [1,8,85,116], ordering constrained models [34,94], and tree-based models [66,86].

In data-to-text generation, Bontcheva and Wilks [10] employed a traditional text generation approach to generate sentences from knowledge base triples in the medical domain. Their model starts with filtering repetitive triples (i.e., content planning) and then group the coherent triples (i.e., document planning). Then, their model aggregates the generated sentences of the coherent triples to produce the final sentences (i.e., surface realization). Cimiano et al. [23] developed a simple model to generate cooking recipes from semantic web data. They focused on using a large corpus to extract lexicon in the cooking domain. The lexicon is then used with a traditional text generation approach to generate cooking recipes. Duma and Klein [37] developed a model to learn a sentence template from a parallel triples-text corpus. Their model first aligns entities in the triples with entities mentioned in sentences. Then, the model extracts templates from the aligned sentences by replacing the entity mentions with unique tokens. These methods employ human-generated rules that work well on triples in a seen domain but fail on triples in an unseen domain.

2.5.2 Neural Text Generation

The encoder-decoder framework [21] is the most commonly used model in neural text generation. The encoder encodes the input (e.g., KB triples) into a vector representation used as a context for the decoder. The decoder generates the output token-by-token constrained by the previously generated token and the context vector computed by the encoder. For entity description generation from knowledge base triples, recent studies use two types of encoders. The first is linear encoder models as in the standard encoder-decoder framework, i.e., treating the input as a sequence. The second is graph-based encoder models that preserve the graph structures of the input triples. We first review the standard encoder-decoder based text generation models. Then, we discuss the graph-based encoder models afterward.

Encoder-decoder Based Text Generation Models

The encoder-decoder framework [21] is a sequence-to-sequence learning model, which is a model that reads the input sequentially and predicts the output token-by-token. This framework has been successfully applied in machine translation to translate a sequence of words from one language to another. Earlier studies adapt the encoder-decoder framework for data-to-text generation by either using simple linear transformation to encode the input or representing the input as a sequence that may discard the input structures. For example, Serban et al. [127] applied the encoder-decoder to generate questions from facts in a KB. In the encoder side, they use a *facts embedding* model that encodes each triple by concatenating vector representation of the subject, the predicate, and the object. In the decoder side, they use *Gated Recurrent Unit* (GRU) [21] and an attention mechanism [4] to generate the question. Wiseman et al. [166] employed the encoder-decoder to generate NBA game summaries. The main challenge handled in their study is to generate a longer text than the typical data-to-text generation models. To handle this problem, they combine the *copy model* [49, 50, 62, 174] and *source reconstruction loss* [149]. The copy model is used to copy words directly from the data-records, e.g., triples. This technique helps to handle the out of vocabulary problem, i.e., unseen entities (words) in the

inference phase. Meanwhile, the source reconstruction loss is used to ensure that the model properly captures the structures of the input. Mei et al. [93] proposed an encoder-aligner-decoder model to generate weather forecasts. The aligner is used to select the most relevant data to be used to predict weather forecasts. These studies experiment on cross-domain datasets for data-to-text generation, and the results show that neural generation models are more flexible to work in an open domain since it is not limited to hand-crafted rules. These results motivate further studies in encoder-decoder based data-to-text generation.

In biography summarization, Lebret et al. [72] developed a conditional neural language model to generate the first sentence of a biography. This model is trained to predict the next word of a sentence not only based on previous words, but also by using features captured from Wikipedia infoboxes. The follow-up studies employ the encoder-decoder framework. Sha et al. [128] proposed a link-based attention model to capture the relationships between properties. Their model consists of three modules, including an encoder, a dispatcher, and a decoder. The encoder and the decoder are similar to the typical data-to-text generation model. Meanwhile, the dispatcher is an improvement over the attention mechanism by integrating a link-based attention model into the content-based attention model. The dispatcher constructs a link matrix that preserves the typical property order in a sentence from a text corpus. Liu et al. [82] proposed a field-gating unit and dual attention mechanism. The field gating unit is an improvement over the LSTM unit by adding a *field gate* into the LSTM unit to update the cell memory based on the corresponding property (i.e., the predicate of the triples). Meanwhile, the dual attention mechanism combines word-level and property-level attentions to capture inter-property relevance.

Another work along this line is the two-stage neural text generation model [112]. First, the two-stage model employs pointer networks [157] to select salient properties and learn its order as a content-plan (i.e., reasonable order of properties in a well-organized sentence). Then it uses the content-plan to generate a summary using the encoder-decoder framework. However, the two-stage models are prone to propagate errors.

These encoder-decoder data-to-text generation models produce fluent text on an or-

dered input. However, these models had decreased performance on disordered input since linearizing the input (i.e., a set of triples) may not yield the proper order of the triples, and hence leads the encoder to produce an improper context to generate a description. To handle this problem, in this thesis, we propose a novel content-plan based attention model that highlights salient properties in a proper order.

Graph-based Text Generation Models

Recent studies propose graph-based encoders to encode the input triples as a graph. Graph-based encoders preserve the input graph structures better than the LSTM encoder that is used in the typical text generation models. This is because the use of the LSTM encoder forces the input to be represented as a sequence that may discard the input structures. Vougiouklis et al. [159] developed *Neural Wikipedian*, which generates a summary from a set of knowledge base triples. First, their model represents each triple as a sequence as opposed to representing the whole triples as a sequence. Then, it uses feed-forward neural networks to encode each triple and concatenate them as the input of the decoder. Marcheggiani and Perez-Beltrachini [89] employed graph convolutional networks (GCN) [68] as the encoder of the input. GCN is used to compute node representation (i.e., entity embeddings) in a graph, based on its graph structures. They use *skip connections* [51,59] to link multiple layers of GCN to get a better node representation. In the decoder side, they use LSTM with an attention mechanism model.

The existing models are good at capturing the relationships between entities in a triple (*intra-triple relationships*). Still, they may fail to capture the relationships between entities in related triples (*inter-triple relationships*). Neural Wikipedian applies feed-forward neural networks for each triple, which makes it only optimized for capturing intra-triple relationships. Meanwhile, GCN may fail to capture long-range relationships between entities since GCN only allows one-hop message passing. In this thesis, we address the aforementioned problem by proposing a supervised topological traversal for a graph-based encoder model. Our graph-based encoder model can handle cycles to capture the intra-triple and inter-triple relationships between entities in a KB.

2.6 Summary

In this chapter, we reviewed the literature related to the problems of knowledge base enrichment as well as the knowledge base development. Previous work tackles the problem of knowledge base enrichment using various approaches such as knowledge bases alignment, relation extraction, and description generation. However, the accuracy and applicability of existing approaches still need improvements. The existing knowledge bases alignment models rely on seed alignments that are rarely available and difficult to obtain due to expensive human efforts required. We address this problem by proposing a novel embedding model that first generates attribute embeddings from the attribute triples and then use this attribute embeddings to shift the entity embeddings of two KBs to the same vector space. This way, entities from different knowledge bases that represent the same real-world entities will have close entity embeddings.

In relation extraction for knowledge base enrichment, the existing models rely on Named Entity Disambiguation (NED) for the canonicalization task. The Open IE methods generate triples where the subject, the predicate, and the object stay in their surface forms. The only means to map the extracted triples to uniquely identified entities in a KB is by post-processing via entity linking methods (i.e., NED). Meanwhile, the entity-aware relation extraction models rely on NED to extract entities in a sentence in a pre-processing step. These methods are prone to error propagation between the extraction and the canonicalization task. To tackle these problems, we propose a novel neural relation extraction model that handles both the extraction and the canonicalization tasks in an end-to-end fashion.

In description generation, the existing encoder-decoder based models represent a graph (i.e., set of triples) as a sequence. These models are unable to handle disordered input since linearizing the input may not yield the proper order of the triples, and hence leads the encoder to produce an improper context to generate a description. Meanwhile, the existing graph-based models good at capturing the relationships between entities in a triple but may fail to capture the relationships between entities in related triples. We handle these problems by proposing an order agnostic graph-based encoder model that uses a supervised topological traversal to encode a graph.

We detail our algorithms to tackle the problem of knowledge base enrichment in the following chapters.

This page intentionally left blank.

Chapter 3

Fully Automatic and Effective Embedding-Based Entity Alignment

This chapter is based on a paper that has been published in the AAAI Conference on Artificial Intelligence 2019: Entity Alignment between Knowledge Graphs Using Attribute Embeddings. Bayu Distiawan Trisedya, Jianzhong Qi, Rui Zhang.

3.1 Introduction

MANY knowledge bases have been created separately in the form of *knowledge graphs* (KGs) for particular purposes. The same entity may exist in different forms in different KGs. For example, `lgd:240111203` in a KG named LinkedGeoData and `dbp:Kromsdorf` in another KG named DBpedia [3] both refer to a village named Kromsdorf in Germany. Typically, these KGs are complementary to each other in terms of completeness. We may integrate such KGs to form a larger KG for knowledge inferences.

In this chapter, we study knowledge bases alignment to enrich a knowledge base by integrating two KGs. Knowledge bases alignment methods aim to find entities in two KBs that represent the same real-world entity, and then integrate these KBs based on the aligned entities. The underlying problem to integrate KGs is identifying the entities in different KGs that represent the same real-world entity, which is commonly referred to as the *entity alignment* problem. We consider KGs where real-world facts are stored in the form of triples. A triple consists of three elements in the form of $\langle subject, predicate, object \rangle$, where *subject* denotes an entity, and *object* denotes either an entity or a literal. Here, if *object* is an entity, the triple is called a *relationship triple*; if *object* is a literal, the triple is called an *attribute triple*. Table 1 gives an example of two subsets of triples from two KGs,

Table 3.1: Knowledge graphs alignment example*

G_1
...
⟨lgd:240111203,geo:long,11.3700843⟩
⟨lgd:240111203,lgd:population,1595⟩
⟨lgd:240111203,rdfs:label,"Kromsdorf"⟩
⟨lgd:240111203,geo:lat,50.9988888889⟩
⟨lgd:240111203,lgd:alderman,"B. Grobe"⟩
⟨lgd:240111203,lgd:is_in,lgd:51477⟩
...
G_2
...
⟨dbp:Kromsdorf,geo:long,11.3701⟩
⟨dbp:Kromsdorf,rdfs:label,"Kromsdorf"⟩
⟨dbp:Kromsdorf,geo:lat,50.9989⟩
⟨dbp:Kromsdorf,dbp:populationTotal,1595⟩
⟨dbp:Kromsdorf,dbp:located_in,dbp:Germany⟩
⟨dbp:Kromsdorf,dbp:district,dbp:Weimarer⟩
...
Merged G_{1_2}
...
⟨lgd:240111203,:long,11.3700843⟩
⟨lgd:240111203,:population,1595⟩
⟨lgd:240111203,:label,"Kromsdorf"⟩
⟨lgd:240111203,:lat,50.9988888889⟩
⟨lgd:240111203,:alderman,"B. Grobe"⟩
⟨lgd:240111203,:is_in,lgd:51477⟩
⟨lgd:240111203,:district,dbp:Weimarer⟩
...

* this example is from our paper: B. D. Trisedya, J. Qi, and R. Zhang, "Entity alignment between knowledge graphs using attribute embeddings," in AAAI, vol. 33, 2019, pp. 297–304.

denoted by G_1 and G_2 (we use prefixes "lgd:" and "dbp:" to simplify the original spell out). The subjects in these two subsets refer to the same entity Kromsdorf, even though they are in different forms "lgd:240111203" and "dbp:Kromsdorf". We aim to identify such entities and give them a unified ID such that both KGs can be merged together through them. In Table 3.1, G_{1_2} denotes the merged KG, where lgd:240111203 is used as the unified ID for the entity Kromsdorf which has a set of properties that is the union

of the sets of properties from both KGs.

Early studies on entity alignment are based on the similarity between the properties of entities [101,110,158]. These methods rely on user-defined rules to determine the properties to be compared between the entities. For example, the properties to be compared between entities of the two KGs in Table 3.1 are `rdfs:label`, `geo:lat`, and `geo:long`. Such approaches are error-prone because different pairs of entities may need different properties to be compared, e.g., for two celebrity entities, they may not have properties such as `geo:lat` and `geo:long`.

Recently, embedding-based models are proposed for entity alignment. Such models are built on top of a KG embedding model, such as translation-based models (e.g., TransE [11]) and Graph Neural Network (GNN-based) models (e.g., GCN [68]). TransE learns *entity embeddings* that capture the similarity between entities within a *single* KG based on the relationship triples in the KG. To adapt the KG embedding for entity alignment between two KGs, the embedding-based models require both predicate and entity embeddings of two KGs to fall in the same vector space. To address this problem, Chen et al. [19,20] and Zhu et al. [186] proposed embedding-based entity alignment models that learn an embedding space for each KG separately and use a *transition matrix* to map the embedding space from one KG to the other as illustrated in Figure 3.1. Their models rely on large numbers of seed alignments (i.e., a seed set of aligned entities from two KGs) to compute the transition matrix. However, the seed alignments between two KGs are rarely available, and hence are difficult to obtain due to expensive human efforts required. Similar to translation-based models, the GNN-based models [14, 143, 163, 168, 172, 175] also require seed alignments. In GNN-based models, the lost function is based on the distances between entities in the seed alignments. Thus, these models have the same problems as the translation-based models when being used for aligning entities from different KGs.

In this thesis, we address the above limitations by proposing *TransAlign*, a fully automatic and effective embedding-based entity alignment model that does not require any human intervention either in aligning predicates or in collecting seed alignments. It is also worth noting that our proposed model does not require seed alignments. TransAlign includes joint learning of entity, predicate, and attribute embeddings to ensure the result-

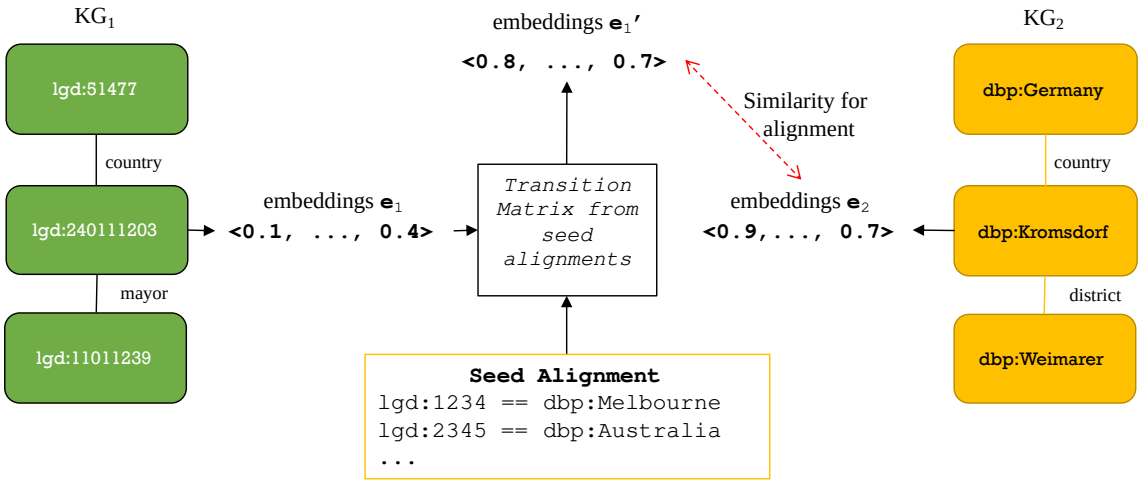


Figure 3.1: Embedding-based knowledge bases alignment

ing embeddings fall in the same vector space. To align predicates from two KGs, we compute predicate embeddings over a predicate proximity graph, where each predicate is a vertex that represents a relationship between entity types (instead of entities). We create such a graph by replacing the subject and object of KG triples by their corresponding types. For example, we replace the triples ⟨dbp:Kromsdorf, dbp:located_in, dbp:Germany⟩ and ⟨lgd:240111203, lgd:is_in, lgd:51477⟩ with the triples ⟨village, dbp:located_in, country⟩ and ⟨village, lgd:is_in, country⟩, respectively. Using the predicate proximity graph, our model can learn the similarity between predicates from two KGs that represent the same relationships, e.g., the predicates dbp:located_in and lgd:is_in.

To yield a unified entity embedding space for two KGs, our model first generates *attribute embeddings* from the attribute triples and then uses this attribute embeddings to shift the entity embeddings of two KGs to the same vector space. We observe that many knowledge graphs contain large numbers of attribute triples, which have not been utilized for entity alignment so far. For example, DBpedia, YAGO, LinkedGeoData, and Geonames contain 47.62%, 62.78%, 94.66%, and 76.78% of attribute triples, respectively. The attribute similarity between two KGs helps the attribute embedding to yield a unified embedding space for two KGs. This enables us to use attribute embeddings to shift the entity embeddings of two KGs into the same vector space and hence allows the en-

tity embeddings to capture the similarity between entities from two KGs. We further use the transitive rule (e.g., by knowing that `Emporium Tower` is located in `London` and `London` is located in `England`, we also know that `Emporium Tower` is located in `England`.) to enrich KG triples for embedding computation.

Our contributions are summarized as follows:

- We propose a fully automatic embedding-based entity alignment model to learn the similarity between entities in two KGs with no seed alignments required (neither predicate nor entity seed alignments).
- To compute the entity embeddings, we propose a novel embedding model that integrates entity embeddings with attribute embeddings to learn a unified embedding space for two KGs.
- We propose a novel fully automatic predicate alignment procedure by learning predicate embeddings from a predicate proximity graph of two KGs to capture the similarity between predicates across two KGs automatically.
- We propose a joint learning scheme of entity, predicate, and attribute embeddings to ensure the resulting embeddings fall in the same vector space.
- We evaluate the proposed model over three real KG pairs. The results show that our model outperforms the state-of-the-art models [14, 19, 141] consistently on the entity alignment task by over 40% in terms of *hits*@1.

The rest of this chapter is organized as follows. Section 3.2 defines the studied problem. Section 3.3 details the proposed model. Section 3.4 presents the experimental results. Section 3.5 summarizes the chapter.

3.2 Preliminary

We start with the problem definition. A knowledge graph G consists of a combination of relationship triples and attribute triples. A relationship triple is in the form of $\langle s, p, o \rangle$,

where p is a relationship (*predicate*) between two entities s (*subject*) and o (*object*). Meanwhile, an attribute triple is in the form of $\langle s, p, a \rangle$ where a is an attribute value of entity s with respect to the predicate (relationship) p. Given two knowledge graphs G_1 and G_2, the task of entity alignment aims to find every pair $\langle s_1, s_2 \rangle$ where $s_1 \in G_1$, $s_2 \in G_2$, and s_1 and s_2 represent the same real-world entity. We use an embedding-based model that assigns a continuous representation for each element of a triple in the forms of $\langle \mathbf{s}, \mathbf{p}, \mathbf{o} \rangle$ and $\langle \mathbf{s}, \mathbf{p}, \mathbf{a} \rangle$, where the bold-face letters denote the vector representations of the corresponding element.

Our proposed model is built on top of a transition-based embedding model such as TransE [11]. We first discuss TransE and its limitations when being used for entity alignment before presenting our proposed model.

3.2.1 TransE

Given a relationship triple $\langle s, p, o \rangle$, TransE suggests that the embeddings of the tail entity o (i.e., object) should be close to the embeddings of the head entity s (i.e., subject) plus the embeddings of the relationship p, i.e., $\mathbf{s} + \mathbf{p} \approx \mathbf{o}$. Such an embedding model aims to preserve the structural information of the entities, i.e., entities that share similar neighbor structures in a knowledge graph should have a close representation in the embedding space. We call it a *structure embedding*. To learn the structure embedding, TransE minimizes a margin-based objective function J_{SE}:

$$J_{SE} = \sum_{t_r \in T_r} \sum_{t'_r \in T'_r} \max\left(0, \left[\gamma + f(t_r) - f(t'_r)\right]\right) \tag{3.1}$$

$$f(t_r) = \|\mathbf{s} + \mathbf{p} - \mathbf{o}\|_2 \tag{3.2}$$

$$T_r = \{\langle s, p, o \rangle | \langle s, p, o \rangle \in G\} \tag{3.3}$$

$$T_r{}' = \left\{\left\langle s', p, o \right\rangle \mid s' \in E\right\} \cup \left\{\left\langle s, p, o' \right\rangle \mid o' \in E\right\} \tag{3.4}$$

Here, $\|\mathbf{x}\|_2$ is the L2-Norm of vector $\mathbf{x}$, γ is a margin hyperparameter, and T_r is the set of valid relationship triples from the training dataset, and T'_r is the set of corrupted relationship triples (E is the set of entities in G). The corrupted triples are used as negative

samples, which are created by replacing the head or tail entity of a valid triple in T_r with a random entity.

TransE has been used to address KG completion tasks that aim to predict missing entities or relations based on existing triples in a KG. TransE constructs a low-dimensional and continuous vector (i.e., embeddings) to describe the latent semantic information (as reflected by the neighboring entities) of a KG. The resulting embeddings capture the similarity between entities in the embedding space. For example, the embedding of `dbp:Germany` should be close to the embedding of `dbp:France` as both entities represent two countries in Europe; they share similar types of neighboring entities (e.g., currency, continent, etc.).

The advantages of structure embedding drive further studies of embedding-based entity alignment. However, a straightforward implementation of structure embedding for entity alignment has limitations: the entity embeddings computed on different KGs may fall in different spaces, where similarity cannot be computed directly. Existing techniques [19, 141, 186] address this limitation by computing a transition matrix to map the embedding spaces of different KGs into the same space, as discussed earlier. However, such techniques require manually collecting a seed set of aligned entities from the different KGs to compute the transition matrix, which do not scale and are vulnerable to the quality of the selected seed aligned entities.

Next, we detail our proposed model to address these limitations.

3.3 Proposed Model

We present an overview of our proposed model in Section 3.3.1. We detail the components of the proposed model afterwards, including *predicate embedding* module in Section 3.3.2, *structure embedding* module in Section 3.3.3, *attribute embedding* module in Section 3.3.4, entity alignment in Section 3.3.6, and triple enrichment in Section 3.3.7.

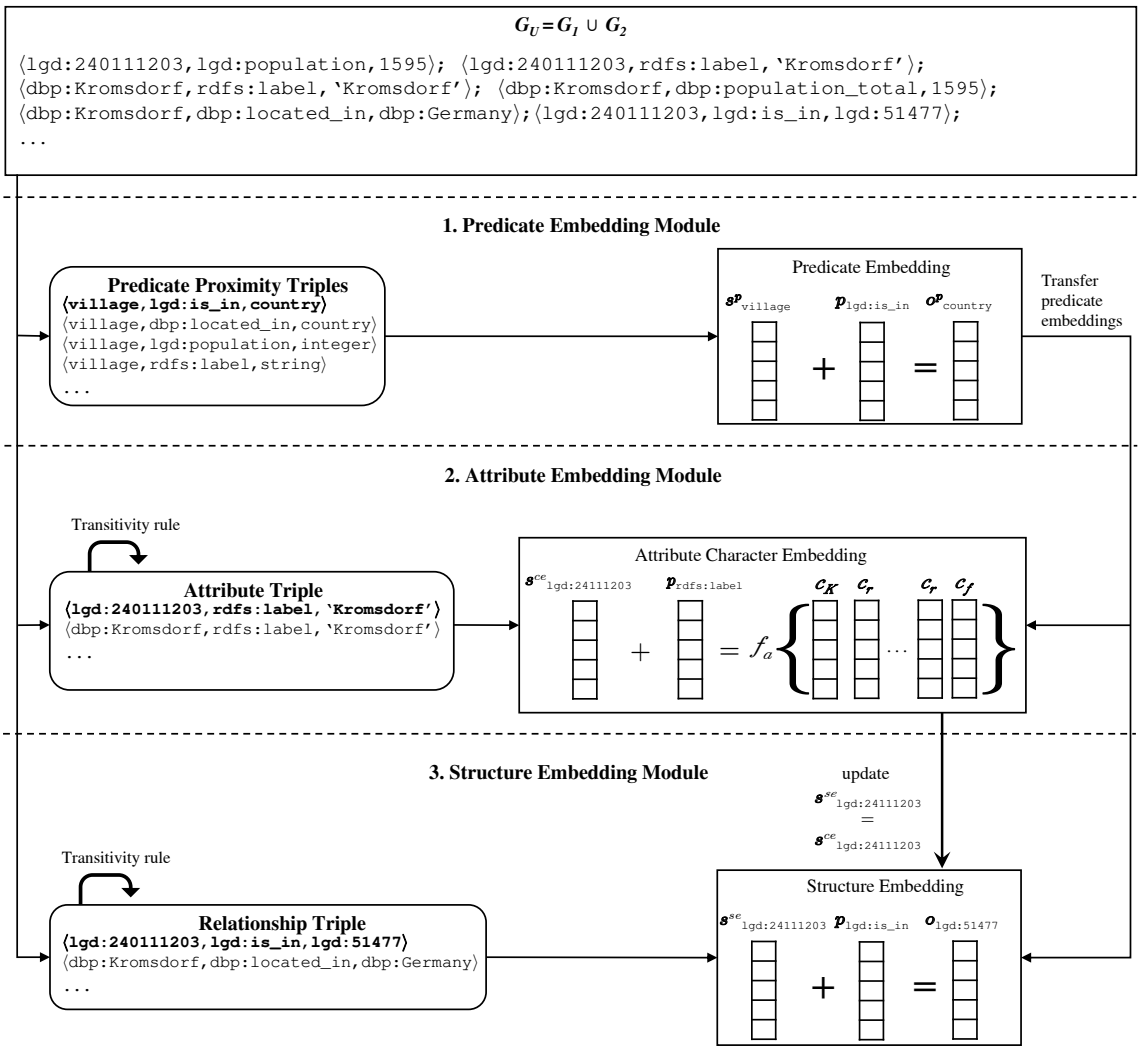


Figure 3.2: Overview of our proposed solution for entity alignment

3.3.1 TransAlign Overview

TransAlign is a fully automatic and effective embedding-based entity alignment model for aligning entities across different KGs. Our proposed model is a joint learning model of entity, predicate, and attribute embeddings from different KGs. TransAlign consists of three embedding modules, including *predicate embedding*, *attribute embedding*, and *structure embedding*. Fig 3.2 illustrates the interaction of these embedding modules in TransAlign.

Embedding-based entity alignment requires the embeddings (both predicate and entity embeddings) of two KGs to fall in the same vector space. To train such a model,

we first combine two knowledge graphs $G_U = G_1 \cup G_2$. From the combined graph, we create three sets of triples, including the set of predicate proximity triples T_p, the set of relationship triples T_r, and the set of attribute triples T_a. In the set of predicate proximity triples, each predicate represents a relationship between entity types. We train the predicate embeddings (detailed in Section 3.3.2) using these triples. This way, our model not only generates a unified embedding space of predicates but also captures the similarity between predicates from two KGs. The predicate embeddings are then used to compute the attribute and the structure embeddings.

The structure embedding (detailed in Section 3.3.3) is learned using the set of relationship triples T_r, while the attribute embedding (detailed in Section 3.3.4) is learned using the set of attribute triples T_a. Initially, the structure embeddings of the entities that come from G_1 and G_2 fall into different vector spaces because the entities from both KGs are represented using different naming schemes. On the contrary, the attribute embeddings learned from the attribute triples T_a can fall into the same vector space. This is achieved by learning character embeddings from the attribute strings, which can be similar even if the attributes are from different KGs (we call it *attribute character embedding*). Then, we use the resulting attribute character embedding to shift the structure embeddings of the entities into the same vector space, which enables the entity embeddings to capture the similarity between entities from two KGs. As an example, suppose that we have triples ⟨`lgd:240111203, lgd:is_in, lgd:51477`⟩ and ⟨`lgd:51477, rdfs:label, "Germany"`⟩ from G_1, and ⟨`dbp:Kromsdorf, dbp:located_in, dbp:Germany`⟩ and ⟨`dbp:Germany, rdfs:label, "Germany"`⟩ from G_2. The attribute character embedding allows both entities `lgd:5147` and `dbp:Germany` to have similar vector representations since both entities have a similar attribute value `"Germany"`. Then, the structure embeddings of entities `lgd:240111203` and `dbp:Kromsdorf` will also be similar since the two entities share similar predicate representation (from the predicate embedding similarity) and have two tail entities `lgd:51477` and `dbp:Germany` which have similar vector representations.

Once we have the embeddings for all entities in G_1 and G_2, the entity alignment module (detailed in Section 3.3.6) finds every pair $\langle s_1, s_2 \rangle$ where $s_1 \in G_1$ and $s_2 \in G_2$ with a

Table 3.2: Example of predicate triples

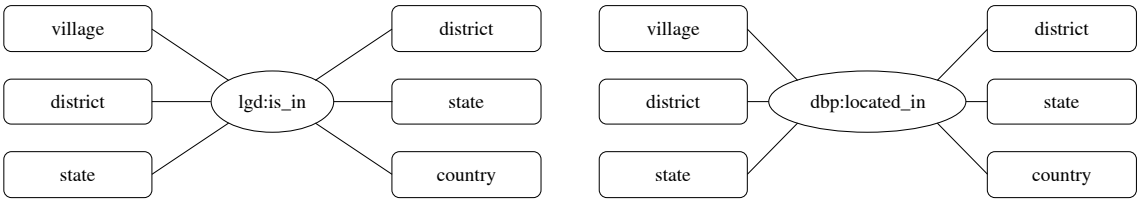
(a) Predicate triples of relationship `lgd:is_in`

LinkedGeoData
...
⟨lgd:240111203, lgd:is_in, lgd:240055406⟩
⟨lgd:240055406, lgd:is_in, lgd:473883922⟩
⟨lgd:473883922, lgd:is_in, lgd:51477⟩
...

(b) Predicate triples of relationship `dbp:located_in`

DBpedia
...
⟨dbp:Kromsdorf, dbp:located_in, dbp:Weimarer_Land⟩
⟨dbp:Weimarer_Land, dbp:located_in, dbp:Thuringia⟩
⟨dbp:Thuringia, dbp:located_in, dbp:Germany⟩
...

(a) Predicate graph of relationship `lgd:is_in` (left) and `dbp:located_in` (right)



(b) Predicate proximity graph of relationship `lgd:is_in` (left) and `dbp:located_in` (right)

Figure 3.3: Graph representation of predicate triples

similarity score above a threshold β.

To further improve the performance of the model, we use the relationship transitivity rule to enrich the properties of an entity that helps build a more robust attribute embedding for computing the similarity between entities. This is detailed in Section 3.3.7.

3.3.2 Predicate Embedding

The same predicates from two knowledge graphs typically connect the same real-world entities. However, they may exist in different surface forms. For the example in Table 3.2, the predicate `lgd:is_in` in LinkedGeoData and the predicate `dbp:located_in` in DBpedia map three entity pairs. In LinkedGeoData, the predicate `lgd:is_in` maps ⟨`lgd:240111203`, `lgd:240055406`⟩, ⟨`lgd:240055406`, `lgd:473883922`⟩, and ⟨`lgd:473883922`, `lgd:51477`⟩. Meanwhile, the predicate `dbp:located_in` in DBpedia maps ⟨`dbp:Kromsdorf`, `dbp:Weimarer_Land`⟩, ⟨`dbp:Weimarer_Land`, `dbp:Thuringia`⟩, and ⟨`dbp:Thuringia`, `dbp:Germany`⟩. Here, the entity pairs from both knowledge graphs correspond to the same real-world entity pairs. For example, the head and tail entities of the entity pair ⟨`lgd:240111203`, `lgd:240055406`⟩ and ⟨`dbp:Kromsdorf`, `dbp:Weimarer_Land`⟩ correspond to a village named Kromsdorf and a district named Weimarer Land, respectively. However, due to different naming schemes of two knowledge graphs, entity embedding models may not capture this similarity. For example, if we applied an entity embedding model to the raw knowledge graph, the predicate embeddings of `lgd:is_in` and `dbp:located_in` may fall in different vector spaces.

To address the above problem and provide a fully automatic predicate alignment procedure, our model learns predicate embeddings from a predicate proximity graph of two KGs. Hence, our model can automatically capture the similarity between predicates across two KGs. A predicate proximity graph is a graph that represents the relationships between entity types instead of entities. We create the predicate proximity graph by replacing entities by their corresponding types. First, we combine two knowledge graphs $G_U = G_1 \cup G_2$. Then, we replace the subject and object of each triple in the combined graph to their corresponding entity types. We take the label of entity types from each knowledge graph to replace the subject and object of the corresponding triples. As illustrated in Fig 3.3(b), we replace the entity `lgd:240111203` with its type, `village`.

We obtain the entity types by extracting the value of `rdfs:type` predicate for all entities from each KG. Typically, each entity has multiple types. For example, the entity `Germany` may have multiple entity types {`thing`, `place`, `location`, `country`} in a KG. Moreover, different KGs may have different schemes of entity types, e.g., in

another KG, the entity `Germany` may have entity types {`place, country`}. To get a consistent entity type scheme from different KGs, we only take the most specific entity type for each entity, e.g., we take `country` instead of `place` for the entity `Germany`. The level of specificity of an entity type is determined by its hierarchy in WordNet [39]. For the attribute values, we extract the corresponding data type for each value (e.g., string, integer, date, etc.).

After obtaining a predicate proximity graph, we compute the predicate embeddings by minimizing the following objective function:

$$J_{PE} = \sum_{t_p \in T_p} \sum_{t'_p \in T'_p} \max\left(0, \left[\gamma + f(t_p) - f(t'_p)\right]\right) \tag{3.5}$$

$$f(t_p) = \left\|\mathbf{s_p} + \mathbf{p} - \mathbf{o_p}\right\|_2 \tag{3.6}$$

$$T_p = \{\langle s_p, p, o_p\rangle | \langle s_p, p, o_p\rangle \in G_p; s_p, o_p \in E_p\} \tag{3.7}$$

$$T_p{}' = \left\{\left\langle s'_p, p, o_p\right\rangle \mid s'_p \in E_p\right\} \cup \left\{\left\langle s_p, p, o'_p\right\rangle \mid_p o' \in E_p\right\} \tag{3.8}$$

Here G_p is the predicate proximity graph, E_p is the set of entities in G_p (i.e., the entity types) T_p and T'_p are the valid and corrupted triples (i.e., for negative samples) from the predicate proximity graph, respectively. By optimizing the above objective function, our model yields a unified predicate embedding space from two knowledge graphs. We maintain the predicates from two knowledge graphs, and hence we can transfer these unified predicate embeddings to learn the structure and the attribute character embeddings.

3.3.3 Structure Embedding

We follow TransE to compute the structure embeddings. TransE considers the same weight for each neighbor in computing the embeddings of an entity, i.e., to update the embeddings of an entity, all neighbors have the same contribution. However, this assumption may not give any advantage in entity alignment since different knowledge graphs are typically do not have the same structure (e.g., have different sets of predicates), which does not help in computing the similarity between entities in different

knowledge graphs. From the example in Table 3.1, the predicates `lgd:alderman` and `dbp:district` do not help in entity alignment.

To tackle the above problem, we adapt TransE to learn the structure embedding for entity alignment between KGs by focusing the embedding learning more on the aligned triples (i.e., triples with aligned predicates). We take benefit from the knowledge base predicate naming scheme convention[1] that provides standardization for typical predicate such as `label, type`, etc. The adaptation is done by adding a weight α to control the embedding learning over the triples. To learn the structure embedding, in our model, we minimize the objective function J_{SE} adapted from Eq. (3.1) as follows:

$$J_{SE} = \sum_{t_r \in T_r} \sum_{t'_r \in T'_r} \max\left(0, \gamma + \alpha\left(f(t_r) - f(t'_r)\right)\right) \tag{3.9}$$

$$\alpha = \frac{count(r)}{|T|} \tag{3.10}$$

where T_r is the set of valid relationship triples, T'_r is the set of corrupted relationship triples, $count(r)$ is the number of occurrences of relationship r, and $|T|$ is the total number of triples in the merge KG G_{1_2}. Typically, the number of occurrences of the aligned predicates is higher than the non-aligned predicates since the aligned predicates appear in both KGs, and hence allows the model to learn more from the aligned triples. For example, for the triples in Table 3.1, the weight α helps the embedding model to focus more on relationships `rdfs:label`, `geo:lat`, and `geo:long` ($\alpha = 2/12$ for each of these predicates) than on relationships `lgd:alderman` or `dbp:district` ($\alpha = 1/12$ for each of these predicates).

3.3.4 Attribute Character Embedding

Following TransE, for the attribute character embedding, we interpret predicate p as a translation from the head entity s to the attribute a. However, the same attribute may appear in different forms in two KGs, e.g., `50.9989` vs. `50.9988888889` as the latitude of an entity; `"Barack Obama"` vs. `"Barack Hussein Obama"` as a person name,

[1] https://www.w3.org/TR/rdf-schema/

etc. Hence, we use a compositional function to encode the attribute value and define the relationship of each element in an attribute triple as $\mathbf{s} + \mathbf{p} \approx f_a(a)$. Here, $f_a(a)$ is a compositional function and a is a sequence of the characters of the attribute value $a = \{c_1, c_2, c_3, ..., c_t\}$. The compositional function encodes the attribute value into a single vector and maps similar attribute values to a similar vector representation. We define three compositional functions as follows.

Sum compositional function (**SUM**). The first compositional function is defined as a summation of all character embeddings of the attribute value.

$$f_a(a) = \mathbf{c_1} + \mathbf{c_2} + \mathbf{c_3} + ... + \mathbf{c_t} \tag{3.11}$$

where $\mathbf{c_1}, \mathbf{c_2}, ..., \mathbf{c_t}$ are the character embeddings of the attribute value. This compositional function is simple, but it suffers in that two strings that contain the same set of characters with a different order will have the same vector representation (i.e., order invariant). For example, two coordinates `"50.15"` and `"15.05"` will have the same vector representation.

LSTM-based compositional function (**LSTM**). To address the problem of SUM, we propose an LSTM-based compositional function. This function uses LSTM networks to encode a sequence of characters into a single vector. We use the final hidden state of the LSTM networks as a vector representation of the attribute value.

$$f_a(a) = f_{lstm}(\mathbf{c_1}, \mathbf{c_2}, \mathbf{c_3}, ..., \mathbf{c_t}) \tag{3.12}$$

where f_{lstm} is the LSTM networks as defined by Hochreiter et al. [55].

N-gram-based compositional function (**N-gram**). LSTM-based compositional function handles the order invariant problem of the SUM compositional function. However, the LSTM-based compositional function only considers the unigram features of a string. To capture rich compositional information of a string, we further propose an N-gram-based compositional function as an alternative to the above compositional functions. Here, we use the summation of the n-gram combination of the attribute value. This way our proposed n-gram compositional function can capture the partial similarity between attribute

values. For example, for two attribute values, `1.7 meter` and `170 cm`, the n-gram compositional function can capture the similarity between the two attribute values based on the characters: `"1"`, `"7"`, and `"m"`.

$$f_a(a) = \sum_{n=1}^{N} \left(\frac{\sum_{i=1}^{l} \sum_{j=i}^{n} \mathbf{c_j}}{t - i - 1} \right) \tag{3.13}$$

where N indicates the maximum value of n used in the n-gram combinations ($N = 10$ in our experiments), and l is the length of the attribute value.

To learn the attribute character embedding, we minimize the following objective function J_{CE}:

$$J_{CE} = \sum_{t_a \in T_a} \sum_{t'_a \in T'_a} \max\left(0, \left[\gamma + \alpha \left(f(t_a) - f(t'_a)\right)\right]\right) \tag{3.14}$$

$$f(t_a) = \|\mathbf{s} + \mathbf{p} - f_a(a)\|_2 \tag{3.15}$$

$$T_a = \{\langle s, p, a\rangle \in G\} \tag{3.16}$$

$$T_a{}' = \left\{ \langle s', p, a\rangle \middle| \, s' \in E \right\} \cup \left\{ \langle s, p, a'\rangle \, | a' \in A \right\} \tag{3.17}$$

Here, T_a is the set of valid attribute triples from the training dataset, while T'_a is the set of corrupted attribute triples (A is the set of attributes in G). The corrupted triples are used as negative samples by replacing the head entity with a random entity or the attribute with a random attribute value. $f(t_a)$ is the plausibility score computed based on the embedding of the head entity s, the embedding of the relationship p, and the vector representation of the attribute value computed using the compositional function $f_a(a)$.

3.3.5 Joint Learning Embedding Model

Our model jointly learns the predicate embeddings, the structure embeddings, and the attribute character embeddings. The proposed model trained over the predicate proximity graph to yield the same predicate embeddings space. Our model uses these predicate embeddings to learn the structure and attribute character embeddings. The attribute embedding model yields the same embeddings space for two knowledge graphs, but lack of structure information. On the other hand, the structure embedding model may yield

different embedding space for two knowledge graphs. To get benefit from the structure embeddings for entity alignment, we use the attribute character embedding $\mathbf{s_{ce}}$ to shift the structure embedding $\mathbf{s_{se}}$ into the same vector space by minimizing the following objective function J_{SIM}:

$$J_{SIM} = \sum_{h \in G_1 \cup G_2} [1 - \cos(\mathbf{s}_{se}, \mathbf{s}_{ce})] \tag{3.18}$$

Here, $\cos(\mathbf{s}_{se}, \mathbf{s}_{ce})$ is the cosine similarity of vector $\mathbf{s}_{se}$ and $\mathbf{s}_{ce}$. As a result, the structure embedding captures the similarity of entities between two KGs based on entity relationships, while the attribute character embedding captures the similarity of entities based on attribute values. The overall objective function of the joint learning of structure embedding and attribute character embedding is:

$$J = J_{PE} + J_{SE} + J_{CE} + J_{SIM} \tag{3.19}$$

3.3.6 Entity Alignment

The existing embedding-based entity alignment models are considered as supervised models that need seed alignments to learn entity alignments from two knowledge graphs. Unlike the existing models, our proposed model is unsupervised – it captures the similarity between entities from two knowledge graphs by learning a unified entity embedding space via predicate and attribute embeddings. Hence, our model does not need seed alignments. Our joint learning embedding model lets similar entities from G_1 and G_2 to have close vector representations. Thus, the resulting embeddings can be used for entity alignment. We compute the following equation for entity alignment.

$$s_{map} = \underset{s_2 \in G_2}{\operatorname{argmax}} \cos(\mathbf{s}_1, \mathbf{s}_2) \tag{3.20}$$

Given an entity $s_1 \in G_1$, we compute the similarity between s_1 and all entities $s_2 \in G_2$; $\langle s_1, s_{map} \rangle$ is the expected pair of aligned entities. We use a similarity threshold β to filter the pair entities that are too dissimilar to be aligned.

3.3.7 Triple Enrichment via Transitivity Rule

In translation based embedding models such as TransE, the embeddings of an entity is learned by aggregating information from its immediate neighbors (i.e., one-hop neighbors). These models may implicitly learn the multi-hop relationships between entities via information propagation after many training epochs. However, the information propagation of the multi-hop relationship is weak. On the other hand, the explicit inclusion of multi-hop relationships (e.g., transitive relationships) increases the number of attributes and related entities for each entity, which helps identify the similarity between entities. For example, given triples $\langle$`dbp:Emporium_Tower, :locatedIn, dbp:London`$\rangle$ and $\langle$`dbp:London, :country, dbp:England`$\rangle$, we can infer that `dbp:Emporium_Tower` has a relationship (i.e., `":locatedInCountry"`) with `dbp:England`. In fact, this information can be used to enrich the related entity `dbp:Emporium_Tower`. We treat the one-hop transitive relation as follows. Given transitive triples $\langle s_1, p_1, o_1 \rangle$ and $\langle o_1, p_2, o_2 \rangle$, we interpret $p_1.p_2$ as a relation from head entity s_1 to tail entity o_2. Therefore, the relationship between these transitive triples is defined as $\mathbf{s_1} + (\mathbf{p_1}.\mathbf{p_2}) \approx \mathbf{o_2}$. The objective functions of the transitivity-enhanced embedding models are adapted from the Eq. (3.9) and Eq. (3.14) by replacing the relationship vector $\mathbf{p}$ with $\mathbf{p_1}.\mathbf{p_2}$.

3.4 Experiments

We design the experiments to show the power of our proposed model from three different aspects. First, we show the power of our model in entity alignment. Second, we show that our predicate embedding model trained over a predicate proximity graph achieves satisfying performance on aligning predicate from different knowledge graphs. Third, we show that the resulting embeddings of our proposed model preserve the structure information of knowledge graphs, which is the original objective of knowledge graph embedding models.

Table 3.3: Statistics of the dataset for entity alignment

Dataset		Entities	Attribute Triples	Relationship Triples
DBP-LGD	LGD	24,309	90,054	10,084
	DBP	22,748	166,008	19,594
DBP-GEO	GEO	21,794	98,790	17,410
	DBP	22,748	166,008	19,594
DBP-YAGO	YAGO	30,628	173,309	38,451
	DBP	33,627	184,672	36,906

3.4.1 Dataset

We evaluate our model on four real KGs including *DBpedia* (DBP) [3], *LinkedGeoData* (LGD) [136], *Geonames* (GEO)[2], and YAGO [57]. We run our proposed model to align entities of DBP with those of LGD, GEO, and YAGO, respectively. We compare the aligned entities found by our model with those in three ground truth datasets, **DBP-LGD**, **DBP-GEO**, and **DBP-YAGO**, which contain aligned entities[3] between DBP and LGD, GEO, and YAGO, respectively. We focus on the `LOCATION` entities in the LGD and GEO KGs since both KGs contain mainly geographical data. We consider `LOCATION`, `PERSON`, and `ORGANIZATION` entities in the YAGO KG. DBP-YAGO contains 15,000 aligned entities and 279 predicates; DBP-LGD contains 10,000 aligned entities and 510 predicates; DBP-GEO contains 10,000 aligned entities and 716 predicates. The datasets are summarized in Table 3.3.

3.4.2 Hyperparameters

We use grid search to find the best hyperparameters for the models. We choose the embeddings dimensionality d among $\{50, 75, 100, 200\}$, the learning rate of the Adam optimizer among $\{0.001, 0.01, 0.1\}$, and the margin γ among $\{1, 5, 10\}$. We train the models with a batch size of 100 and a maximum of 400 epochs.

[2]http://www.geonames.org/ontology/
[3]http://downloads.dbpedia.org/2016-10/links/

3.4.3 Compared Models

We compare TransAlign with four existing embedding-based entity alignment models, including:

- **TransE** [11], which is the adapted knowledge graph embedding model for entity alignment. We use the same procedure as in our proposed model to perform entity alignment using TransE. First, we combine two knowledge graphs. Then, we apply TransE to compute the entity embeddings of the combined graph. Finally, we compare the embeddings for entity alignment.

- **MTransE** [19], which is the state of the art embedding-based entity alignment model on top of TransE. MTransE learns a transition matrix from seed alignments to yield a unified embedding space from two knowledge graphs.

- **JAPE** [141], which is another state of the art embedding-based entity alignment model on top of TransE. JAPE combines the relationship triples with masked attribute triples. A masked attribute triple is an attribute triple in which its object (literals) is replaced by its data type (e.g., string, integer, etc.).

- **MuGNN** [14], which is the state-of-the-art embedding-based model on top of Graph Convolutional Network (GCN) [68].

3.4.4 Entity Alignment Results

We evaluate the performance of the models using **hits@k**$(k = 1, 10)$ (i.e., the proportion of correctly aligned entities ranked in the top k predictions), and the mean of the rank (**MeanRank**) of the correct (i.e., matching) entity. Higher *hits@k* and lower MeanRank indicate better performance. For each entity from DBP, we use Eq. (3.20) to compute the similarity scores with entities from the other KG (LGD/GEO/YAGO).

Table 3.4 shows that our proposed model consistently outperforms the baseline models, with $p < 0.01$ based on the t-test on the MR. As expected, TransE has poor performance on the entity alignment task because its embeddings of different KGs fall into different vector spaces. Hence, it fails to capture the entity similarity between KGs.

Table 3.4: Performance comparisons of entity alignment models

(a) Entity alignment on LGD-DBP dataset

Model	LGD-DBP		
	hits@1	hits@10	MeanRank
TransE	2.61	7.01	18445
MTransE	33.29	34.32	10194
JAPE	33.33	33.35	5104
MuGNN	38.62	39.97	3526
SUM	50.26	63.11	482
LSTM	59.11	71.27	302
N-gram	82.96	**92.54**	35
+Transitivity			
SUM	51.88	65.61	392
LSTM	60.97	72.96	126
N-gram	**84.17**	92.05	**29**

(b) Entity alignment on GEO-DBP dataset

Model	GEO-DBP		
	hits@1	hits@10	MeanRank
TransE	1.34	6.71	17145
MTransE	33.34	33.98	10240
JAPE	33.35	33.75	5088
MuGNN	37.01	41.22	3546
SUM	50.23	62.81	915
LSTM	61.87	73.08	208
N-gram	85.31	92.72	78
+Transitivity			
SUM	52.28	64.39	816
LSTM	62.35	72.91	259
N-gram	**86.91**	**93.32**	**23**

(c) Entity alignment on YAGO-DBP dataset

Model	YAGO-DBP		
	hits@1	hits@10	MeanRank
TransE	1.22	3.54	24809
MTransE	33.46	34.32	7105
JAPE	33.35	33.37	5296
MuGNN	39.32	41.87	3987
SUM	81.14	81.72	158
LSTM	86.14	92.18	104
N-gram	90.31	94.43	21
+Transitivity			
SUM	80.78	82.14	209
LSTM	86.88	91.07	75
N-gram	**91.44**	**94.52**	**19**

Meanwhile, MTransE, JAPE, and MuGNN rely on the number of the seed alignments, i.e., the larger the number of the seed alignment, the better their performances. In this experiment, we take 30% of the gold standard as the seed alignments, as suggested in the original papers.

To further investigate the effect of seed alignments on the existing methods, we gradually increase the proportion of seed alignments, from 10% to 50%. The results in Fig. 3.4

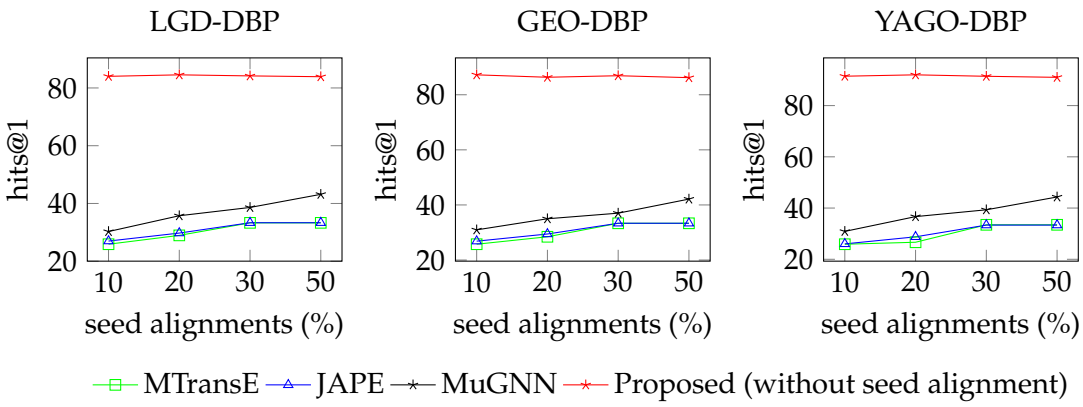


Figure 3.4: The effect of seed alignment size on the existing embedding-based entity alignment models.

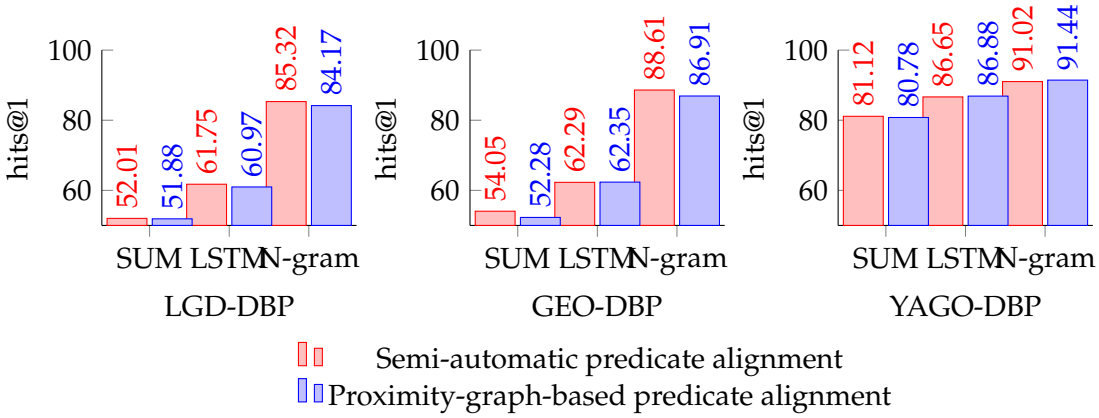


Figure 3.5: The effect of predicate alignment models.

show that the performances of these methods are increasing along with the increase of the seed alignment proportions, i.e., these methods rely on seed alignments. In contrast, our proposed method does not need any seed alignment. The predicate alignments are learned from the predicate proximity graph, while the entity alignments are learned from the combination of the structure embeddings and the attribute character embeddings.

Among our attribute character embedding models, the N-gram compositional function gives the best performance. This is because the N-gram compositional function preserves string similarity better when mapping attribute strings to their vector representations than the other functions. The transitivity rule further improves the performance of the model since it enriches the attributes of the entities, which allows more attributes to

be used in the alignment.

3.4.5 Predicate Alignment Methods Comparison

To show the power of a predicate proximity graph for predicate alignment, we compare it with a semi-automatic predicate alignment procedure. In the semi-automatic predicate alignment procedure, we rename the similar predicates of two KGs with a unified naming scheme to have a unified vector space for the relationship embeddings. First, we use string edit distance to find the similar predicates from two KGs, and then we manually remove the false positive. From the results in Fig. 3.5, we can see that both approaches (i.e., the predicate proximity graph and the semi-automatic predicate alignment) help our entity alignment model achieve comparable performance in terms of *hits*@1, while our proposed predicate embeddings method does not require any error-prone manual intervention.

3.4.6 Discussion

To evaluate the power of our attribute character embedding in capturing the similarity between entities, we further create rule-based models for entity alignment, where we simply use the edit distance between entity label strings to align the entities. For the DBP-LGD and DBP-GEO datasets, we add coordinate similarity as an additional feature since both datasets only contain `LOCATION` entities. From Table 3.5, we can see that the resulting embeddings of our model can be added as another additional feature to enhance the performance of the rule-based models.

We further evaluate our proposed model on KG completion tasks. We follow two standard tasks, including link prediction [11] and triple classification [134]. Link prediction aims to predict the missing element (either the entity s or o, i.e., *entity link prediction*, or the predicate p, i.e., *predicate link prediction*) given two other elements (i.e., predicting s given p and o; predicting o given s and p; or predicting p given s and o). The evaluation protocol for link prediction is defined as follows. First, each relationship triple is corrupted by replacing one of its elements (i.e., head, predicate, or tail) with all possible elements in the dataset. Then, these corrupted triples are ranked ascendingly based

Table 3.5: Rule-based entity alignment results

(a) Rule-based entity alignment on LGD-DBP dataset

Model	LGD-DBP		
	hits@1	hits@10	MeanRank
String	80.52	80.57	2603
String + Coord	86.27	88.85	380
String (+ Coord) + Embeddings	**88.25**	**89.04**	**36**

(b) Rule-based entity alignment on GEO-DBP dataset

Model	GEO-DBP		
	hits@1	hits@10	MeanRank
String	79.32	79.41	2048
String + Coord	87.61	89.15	441
String (+ Coord) + Embeddings	**91.27**	**90.89**	**29**

(c) Rule-based entity alignment on YAGO-DBP dataset

Model	YAGO-DBP		
	hits@1	hits@10	MeanRank
String	76.41	76.88	484
String + Coord	n/a	n/a	n/a
String (+ Coord) + Embeddings	**86.22**	**86.51**	**69**

on the plausibility score $(\mathbf{s} + \mathbf{p} - \mathbf{o})$ (i.e., valid triples should have smaller plausibility scores than corrupted triples). We report *hits*@10 for the link prediction task. Triple classification aims to determine whether a triple $\langle s, p, o \rangle$ is a valid relationship triple or not. A binary classifier is trained based on the plausibility score $(\mathbf{s} + \mathbf{p} - \mathbf{o})$. We report the percentage of correctly classified triples.

Table 3.6 shows the results of KG completion tasks. Despite not being specifically designed for KG completion tasks, our proposed model achieves competitive performance on these tasks compared to TransE, which was proposed for these tasks. This degradation in performance is due to that parameter α in our model guides the model to learn more on the aligned triples. However, the degradation is not significant with $p > 0.05$ based on

Table 3.6: Knowledge Graph Completion Results

	Entty Link Prediction			Predicate Link Prediction			Triple Classification		
	hits@10			hits@10			Precision		
	LGD	GEO	YAGO	LGD	GEO	YAGO	LGD	GEO	YAGO
TransE	78.80	78.77	65.81	86.06	86.29	86.62	80.46	76.94	66.45
MTransE	65.55	63.89	60.40	80.95	80.98	81.00	77.41	73.81	63.21
JAPE	72.89	71.97	61.31	82.64	82.67	82.87	75.19	72.94	62.32
SUM	76.98	74.89	63.11	84.64	84.53	83.43	78.16	73.25	63.12
LSTM	75.17	75.11	63.43	84.31	84.05	84.04	79.21	74.16	63.74
N-gram	75.82	76.75	63.65	83.98	83.85	83.76	77.98	74.31	64.15
Transitivity-enhanced model									
SUM	73.11	72.87	61.34	83.74	83.49	83.34	77.86	73.12	62.25
LSTM	74.56	74.47	61.99	83.12	83.06	82.90	77.98	72.19	64.17
N-gram	74.98	75.85	63.19	82.76	82.31	82.02	79.21	73.97	63.81

the t-test on the MR of the link prediction results. Moreover, our method achieves better performance than the existing embedding-based entity alignment models MTransE and JAPE on these tasks.

3.5 Summary

In this chapter, we studied the problem of knowledge bases alignment for enriching a knowledge base and propose an embedding model for aligning entities from two different KBs. Our proposed model integrates predicate embedding, structure embedding, and attribute character embedding. The predicate embedding is computed over a predicate proximity graph to ensure that the predicate embeddings of two KGs have a unified embedding space. Meanwhile, to ensure a unified entity embedding space, our proposed model uses the attribute character embedding to shift the entity embeddings from different KGs to the same vector space. Moreover, we adopt the transitivity rule to enrich the number of attributes of an entity that helps identify the similarity between entities based on the attribute embeddings.The experimental results show that our proposed model outperforms the baselines consistently. The results on knowledge base completion show

that our proposed models that jointly learns the entity, predicate, and attribute embeddings can capture the similarity between entities and predicates in both within a KG and across KGs. The alignment accuracy of our model measured by *hits*@1 is consistently over 80%, which makes our proposed model more *practical* to be used than the existing models.

This page intentionally left blank.

Chapter 4

An End-to-end Relation Extraction and Canonicalization Model for Knowledge Base Enrichment

This chapter is based on a paper that has been published in the Annual Meeting of the Association for Computational Linguistics (ACL) 2019: Neural Relation Extraction for Knowledge Base Enrichment. Bayu Distiawan Trisedya, Gerhard Weikum, Jianzhong Qi, Rui Zhang.

4.1 Introduction

KNOWLEDGE bases alignment (KBA) is one of the approaches to enrich a knowledge base. We have detailed our work on knowledge bases alignment in Chapter 3. Knowledge base enrichment via KBA is done by first aligning entities from different knowledge bases that represent the same real-world entity. Then, the properties of the aligned entities are merged to form a more extensive knowledge base. However, a merged KB from large knowledge bases such as DBpedia [3], Wikidata [160], and Yago [139] are far from complete and mandate continuous enrichment and curation.

In this chapter, we present another approach to enrich a knowledge base via relation extraction from textual sources. Specifically, we aim to extract triples in the form of $\langle s, p, o \rangle$, where s is a head entity (i.e., subject), o is a tail entity (i.e., object), and p is a relationship (i.e., predicate) between the entities. Importantly, as KBs typically have much better coverage on entities than on relationships, we assume that s and o are existing entities in a KB, p is a predicate that falls in a predefined set of predicates that we are interested in, but the relationship triple $\langle s, p, o \rangle$ does not exist in the KB yet. We

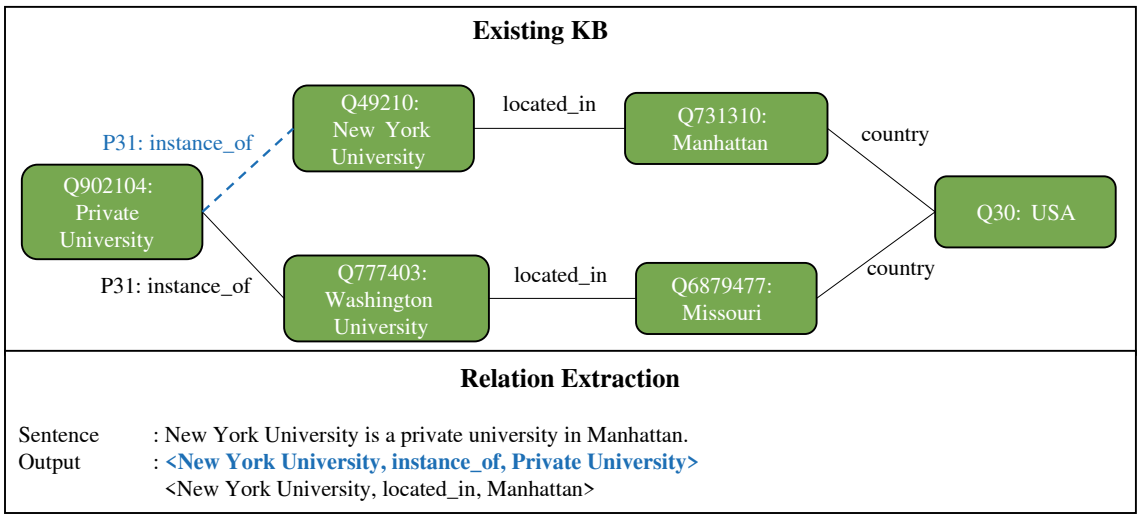


Figure 4.1: Relation extraction for knowledge base enrichment

aim to find more relationships between s and o and add them to the KB. For example, from the first extracted triples in Figure 4.1, we may recognize two entities `New York University` and `Private University`, which already exist in the KB; also the predicate `instance_of` is in the set of predefined predicates we are interested in, but the relationship triple ⟨`New York University, instance_of, Private University`⟩ does not exist in the KB. We aim to add this relationship to our KB (indicated by the blue dashed line in the graph in Figure 4.1). This is a typical situation for KB enrichment (as opposed to constructing a KB from scratch or performing relation extraction for other purposes, such as question answering or summarization).

KB enrichment mandates that the entities and relationships of the extracted triples are canonicalized by mapping them to their proper entity and predicate IDs in a KB. Table 4.1 illustrates an example of triples extracted from a sentence. The entities and predicate of the first extracted triple, including `New York University`, `instance_of`, and `Private University`, are mapped to their unique IDs `Q49210`, `P31`, and `Q902104`, respectively, to comply with the semantic space of the KB.

Previous studies on relation extraction have employed both unsupervised and supervised approaches. Unsupervised approaches typically start with a small set of manually defined extraction patterns to detect entity names and phrases about relationships in an

Table 4.1: Example of relation extraction from a sentence

Input sentence:
"New York University is a private university in Manhattan."
Unsupervised approach output:
⟨New York University,is,private university⟩ ⟨New York University,is private university in,Manhattan⟩
Supervised approach output:
⟨New York University, instance_of, Private University⟩ ⟨New York University, located_in, Manhattan⟩
Canonicalized output:
⟨Q49210, P31, Q902104⟩ ⟨Q49210, P131, Q11299⟩

input text. This paradigm is known as *Open Information Extraction* (Open IE) [5,27,47]. In this line of approaches, both entities and predicates are captured in their surface forms without canonicalization. Supervised approaches train statistical and neural models for inferring relationships between two known entities in a sentence [80, 97, 117, 118, 176]. Most of these studies employ pre-processing steps to recognize entities. Only a few studies have integrated the mapping of extracted triples onto uniquely identified KB entities by using logical reasoning on the existing KB to disambiguate the extracted entities [122,140].

Most existing methods thus entail the need for *Named Entity Disambiguation* (NED) (cf. the survey by Shen et al. [130]) as a separate processing step. In addition, the mapping of relationship phrases onto KB predicates necessitates another mapping step, typically aided by paraphrase dictionaries. This two-stage architecture is inherently prone to error propagation across its two stages: NED errors may cause extraction errors (and vice versa) that lead to inaccurate relationships being added to the KB.

We aim to integrate the extraction and the canonicalization tasks by proposing an end-to-end neural learning model to jointly extract triples from sentences and map them into an existing KB. Our method is based on the encoder-decoder framework [21] by treating the task as a translation of a sentence into a sequence of elements of triples. For the example in Table 4.1, our model aims to translate the sentence "New York University

`is a private university in Manhattan`" into a sequence of IDs "`Q49210 P31 Q902104 Q49210 P131 Q11299`", from which we can derive two triples ⟨`Q49210, P31, Q902104`⟩ and ⟨`Q49210, P131, Q11299`⟩ to be added to the KB.

A standard encoder-decoder model with attention [4] is, however, unable to capture the multi-word entity names and verbal or noun phrases that denote predicates. To address this problem, we propose a novel form of n-gram based attention that computes the n-gram combination of attention weight to capture the verbal or noun phrase context that complements the word level attention of the standard attention model. Our model thus, can better capture the multi-word context of entities and relationships. Our model harnesses pre-trained word and entity embeddings that are jointly learned with skip-gram [95] and TransE [11]. The advantages of our jointly learned embeddings are twofold. First, the embeddings capture the relationship between words and entities, which is essential for named entity disambiguation. Second, the entity embeddings preserve the relationships between entities, which help to build a highly accurate classifier to filter the invalid extracted triples. To cope with the lack of fully labeled training data, we adapt distant supervision to generate aligned pairs of sentence-triple as the training data. We augment the process with co-reference resolution [25] and dictionary-based paraphrase detection [44, 48]. Co-reference resolution helps extract sentences with implicit entity names, which enlarges the set of candidate sentences to be aligned with existing triples in a KB. Meanwhile, paraphrase detection helps filter sentences that do not express any relationships between entities.

Our contributions are summarized as follows:

- We propose an end-to-end model for extracting and canonicalizing triples to enrich a KB. The model reduces error propagation between relation extraction and NED, which existing approaches are prone to.
- We propose an n-gram based attention model to effectively map the multi-word mentions of entities and their relationships into uniquely identified entities and predicates. We propose joint learning of word and entity embeddings to capture the relationship between words and entities for named entity disambiguation. We further propose a modified beam search and a triple classifier to generate high-

quality triples.

- We evaluate the proposed model over two real-world datasets. We adapt distant supervision with co-reference resolution and paraphrase detection to obtain high-quality training data. The experimental results show that our model consistently outperforms a strong baseline for neural relation extraction [80] coupled with state-of-the-art NED models [56, 70].

The rest of this chapter is organized as follows. Section 4.2 explains the proposed model. Section 4.3 presents the experimental results. Section 4.4 summarizes the chapter.

4.2 Proposed Model

We start with the problem definition. Let $G = (E, R)$ be an existing KG where E and R are the sets of entities and relationships (predicates) in G, respectively. We consider a sentence $S = \langle x_1, x_2, ..., x_i \rangle$ as the input, where x_i is a token at position i in the sentence. We aim to extract a set of triples $Y = \{y_1, y_2, ..., y_j\}$ from the sentence, where $y_j = \langle s_j, p_j, o_j \rangle$, $s_j, o_j \in E$, and $p_j \in R$. Table 4.1 illustrates the input and target output of our problem.

4.2.1 Solution Framework

Figure 4.2 illustrates the overall framework. Our framework consists of three components: *data collection* module, *embedding* module, and *neural relation extraction* module.

In the data collection module (detailed in Section 4.2.2), we align known triples in an existing KB with sentences that contain such triples from a text corpus. The aligned pairs of sentences and triples will later be used as the training data in our neural relation extraction module. This alignment is done by distant supervision. To obtain a large number of high-quality alignments, we augment the process with a co-reference resolution procedure to extract sentences with implicit entity names, which enlarges the set of candidate sentences to be aligned. We further use dictionary-based paraphrase detection to filter sentences that do not express any relationships between entities.

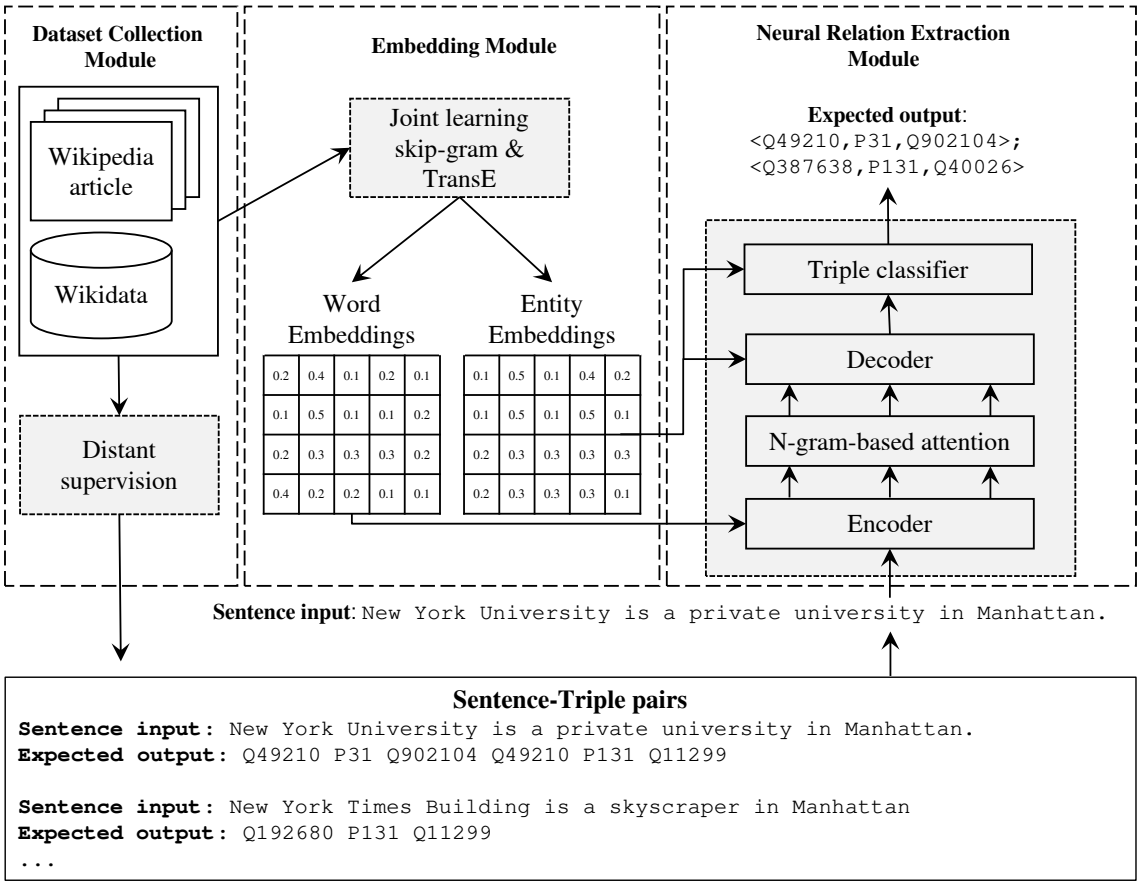


Figure 4.2: Proposed solution for relation extraction to enrich a knowledge base

In the embedding module (detailed in Section 4.2.3), we propose joint learning of word and entity embeddings by combining skip-gram [95] to compute the word embeddings and TransE [11] to compute the entity embeddings. The objective of the joint learning model is to capture the similarity of words and entities that helps map the entity names into the related entity IDs. Moreover, the resulting entity embeddings are used to train a triple classifier that helps filter invalid triples generated by our relation extraction model.

In the neural relation extraction module (detailed in Section 4.2.4), we propose an n-gram based attention model by expanding the attention mechanism to the n-gram token of a sentence. The n-gram attention computes the n-gram combination of attention weight to capture the verbal or noun phrase context that complements the word level

attention of the standard attention model. This expansion helps our model better in capturing the multi-word context of entities and relationships.

The output of the encoder-decoder model is a sequence of the entity and predicate IDs where every three IDs indicate a triple. To generate high-quality triples, we propose two strategies. The first strategy uses a modified beam search that computes the lexical similarity of the extracted entities with the surface form of entity names in the input sentence to ensure the correct entity prediction. The second strategy uses a triple classifier that is trained using the entity embeddings from the joint learning to filter the invalid triples. The triple generation process is detailed in Section 4.2.5

4.2.2 Dataset Collection

We aim to extract triples from a sentence for KB enrichment by proposing a supervised relation extraction model. To train such a model, we need a large volume of labeled training data in the form of sentence-triple pairs. Following Sorokin et al. [135], we use distant supervision [97] to align sentences in Wikipedia[1] with triples in Wikidata[2] [160].

Figure 4.3 illustrates the dataset collection process. We map an entity mention in a sentence to the corresponding entity entry (i.e., Wikidata ID) in Wikidata via the hyperlink associated with the entity mention, which is recorded in Wikidata as the `URL` property of the entity entry. Each pair may contain one sentence and multiple triples. We sort the order of the triples based on the order of the predicate paraphrases that indicate the relationships between entities in the sentence. We collect sentence-triple pairs by extracting sentences that contain both head and tail entities of Wikidata triples.

To generate high-quality sentence-triple pairs, we propose two additional steps: (1) extracting sentences that contain implicit entity names using co-reference resolution, and (2) filtering sentences that do not express any relationships using paraphrase detection. We detail these steps as follows. Prior to aligning the sentences with triples, in Step (1), we find the implicit entity names to increase the number of candidate sentences to be aligned. We apply co-reference resolution [25] to each paragraph in a Wikipedia

[1]https://dumps.wikimedia.org/enwiki/latest/enwiki-latest-pages-articles.xml.bz2
[2]https://dumps.wikimedia.org/wikidatawiki/entities/latest-all.ttl.gz

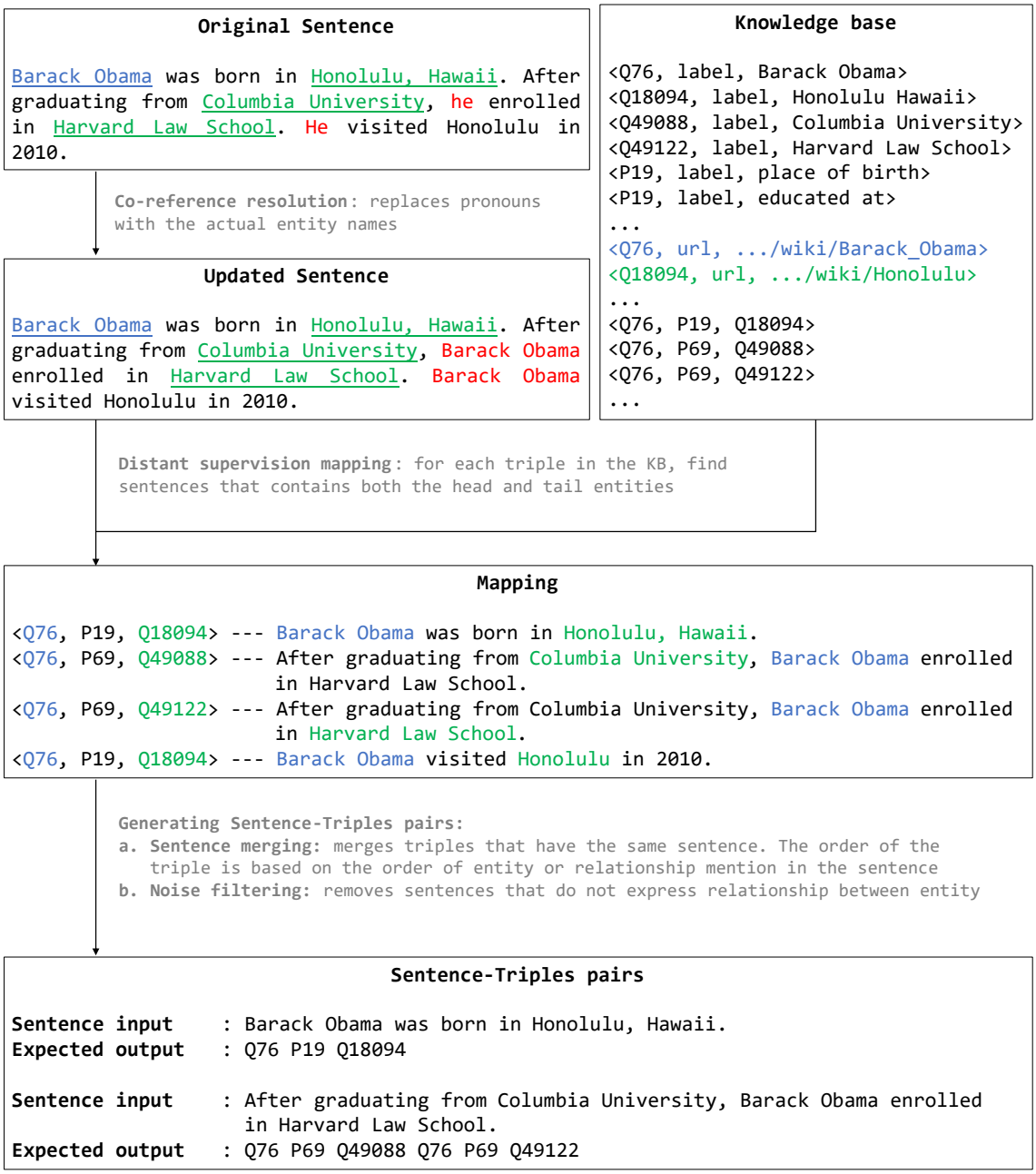


Figure 4.3: Dataset collection for relation extraction

article and replace the extracted co-references with their actual entity names. We observe that the first sentence of a paragraph in a Wikipedia article may contain a pronoun that refers to the main entity. For example, there is a paragraph on the Wikipedia page of entity `Barack Obama` that starts with a sentence "`He was re-elected to the`

Illinois Senate in 1998". This may cause the co-reference resolution to lose the implicit entity names for the rest of the paragraph. To address this problem, we heuristically replace the pronouns in the first sentence of a paragraph if the main entity name of the Wikipedia page is not mentioned. For the sentence in the previous example, we replace "He" with "Barack Obama". The intuition is that a Wikipedia article contains content of a single entity of interest and that the pronouns mentioned in the first sentence of a paragraph mostly relate to the main entity.

In Step (2), we use dictionary-based paraphrase detection to capture relationships between entities in a sentence. First, we create a dictionary by populating predicate paraphrases from three sources, including PATTY [99], POLY [48], and PPDB [44] that yield 540 predicates and 24,013 unique paraphrases. For example, predicate paraphrases for the relationship "place of birth" are {born in, was born in, ...}. Then, we use this dictionary to filter sentences that do not express any relationships between entities. We use exact string matching to find verbal or noun phrases in a sentence, which is paraphrases of a predicate of a triple. For example, for the triple ⟨Barack Obama, place of birth, Honolulu⟩, the sentence "Barack Obama was born in 1961 in Honolulu, Hawaii" will be retained while the sentence "Barack Obama visited Honolulu in 2010" will be removed (the sentence may be retained if there is another valid triple ⟨Barack Obama, visited, Honolulu⟩). This helps filter noises for the sentence-triple alignment.

The above strategies are used to obtain extensive training data automatically. The use of co-reference resolution is to obtain more sentences that are not extracted using the distance supervision technique. Meanwhile, filtering using paraphrase detection is used to remove sentences that do not express any relationships. We use three paraphrase dictionaries, which are the most representative dictionary available. These dictionaries enable us to automatically obtain extensive training data rather than manually annotate the data, which is very costly.

The collected dataset contains 255,654 sentence-triple pairs. For each pair, the maximum number of triples is four (i.e., a sentence can produce at most four triples). We split the dataset into a train set (80%), a dev set (10%), and a test set (10%) (we call it the **WIKI**

Table 4.2: Statistics of the dataset for relation extraction

	#pairs	#triples	#entities	#predicates
All (WIKI)	255,654	330,005	279,888	158
Train+val	225,869	291,352	249,272	157
Test (WIKI)	29,785	38,653	38,690	109
Test (GEO)	1,000	1,095	124	11

test dataset). For *stress testing* (to test the proposed model on a different style of text than the training data), we also collect another test dataset outside Wikipedia. We apply the same procedure to the user reviews of a travel website. First, we collect user reviews on 100 famous landmarks in Australia. Then, we apply the adapted distant supervision to the reviews and collect 1,000 sentence-triple pairs (we call it the **GEO** test dataset). Table 4.2 summarizes the statistics of our datasets.

4.2.3 Joint Learning of Word and Entity Embeddings

Our relation extraction model is based on the encoder-decoder framework, which has been widely used in Neural Machine Translation to translate text from one language to another. In our setup, we aim to translate a sentence into triples, and hence the vocabulary of the source input is a set of English words, while the vocabulary of the target output is a set of entity and predicate IDs in an existing KG. To compute the embeddings of the source and target vocabularies, we propose joint learning of word and entity embeddings that is effective in capturing the similarity between words and entities for named entity disambiguation [173]. Note that our method differs from that of Yamada et al. [173]. We use joint learning by combining skip-gram [95] to compute the word embeddings and TransE [11] to compute the entity embeddings (including the relationship embeddings). In contrast, Yamada et al. [173] use Wikipedia Link-based Measure (WLM) [96] that does not consider the relationship embeddings.

Our model learns the entity embeddings by minimizing a margin-based objective

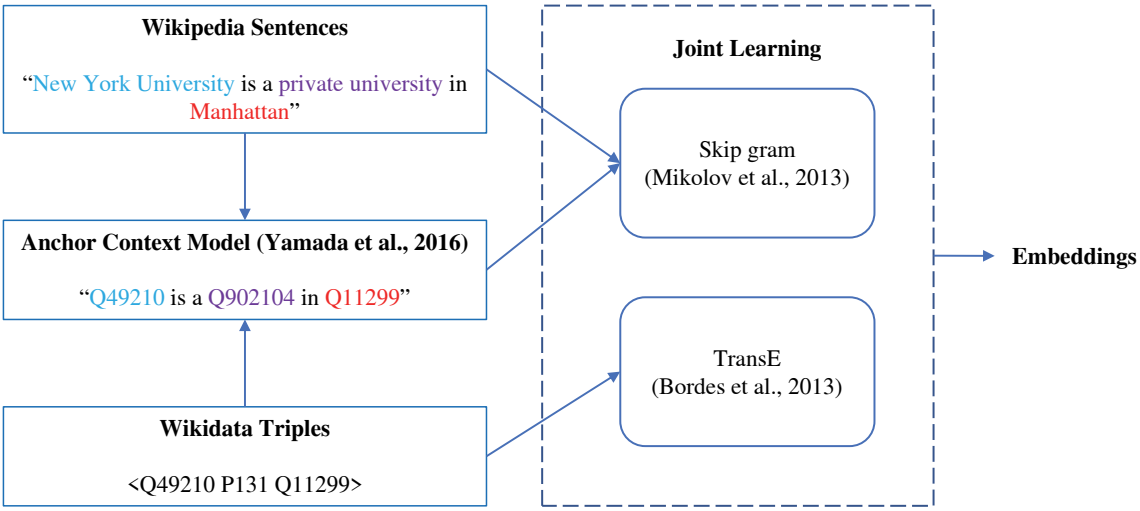


Figure 4.4: Joint learning of word and entity embeddings for neural relation extraction

function J_E:

$$J_E = \sum_{t_r \in T_r} \sum_{t'_r \in T'_r} \max\left(0, \left[\gamma + f(t_r) - f(t'_r)\right]\right) \tag{4.1}$$

$$T_r = \{\langle s, p, o\rangle | \langle s, p, o\rangle \in G\} \tag{4.2}$$

$$T_r{}' = \left\{\langle s', p, o\rangle \mid s' \in E\right\} \cup \left\{\langle s, p, o'\rangle \mid o' \in E\right\} \tag{4.3}$$

$$f(t_r) = \|\mathbf{s} + \mathbf{p} - \mathbf{o}\| \tag{4.4}$$

Here, $\|\mathbf{x}\|$ is the L1-Norm of vector $\mathbf{x}$, γ is a margin hyperparameter, T_r is the set of valid relationship triples from a KG G, and T'_r is the set of corrupted relationship triples (recall that E is the set of entities in G). The corrupted triples are used as negative samples, which are created by replacing the head or tail entity of a valid triple in T_r with a random entity. We use all triples in Wikidata except those which belong to the testing data to compute the entity embeddings.

To establish the interaction between the entity and word embeddings, we follow the *Anchor Context Model* proposed by Yamada et al. [173]. First, we generate a text corpus by combining the original text and the modified anchor text of Wikipedia. This is done by replacing the entity names in a sentence with the related entity or predicate IDs. For example, the sentence "`New York University is a private university in`

Manhattan" is modified into "Q49210 is a Q902104 in Q11299". Then, we use the skip-gram method to compute the word embeddings from the generated corpus (the entity IDs in the modified anchor text are treated as words in the skip-gram model). Figure 4.4 illustrates the joint learning embedding model. Given a sequence of n words $[x_1, x_2, ..., x_n]$, the model learns the word embeddings, by minimizing the following objective function J_W:

$$J_W = \frac{1}{T} \sum_{t=1}^{n} \sum_{-c \leq j \leq c, j \neq 0} \log P\left(x_{t+j} | x_t\right) \tag{4.5}$$

$$P\left(x_{t+j} | x_t\right) = \frac{\exp(\mathbf{v}_{x_{t+j}}^{'\top} \mathbf{v}_{x_t})}{\sum_{i=1}^{X} \exp(\mathbf{v}_i^{'\top} \mathbf{v}_{x_t})} \tag{4.6}$$

where c is the size of the context window, x_t denotes the target word, and x_{t+j} is the context word; $\mathbf{v}_x$ and $\mathbf{v}_x^{'}$ are the input and output vector representations of word x, and X is the vocabulary size. The overall objective function of the joint learning of word and entity embeddings is:

$$J = J_E + J_W \tag{4.7}$$

4.2.4 N-gram Based Attention Model

Our proposed relation extraction model integrates the extraction and canonicalization tasks for KB enrichment in an end-to-end manner. To build such a model, we employ an encoder-decoder model [21] to translate a sentence into a sequence of triples. The encoder encodes a sentence into a vector that is used by the decoder as a context to generate a sequence of triples. Because we treat the input and output as a sequence, We use the LSTM networks [55] in the encoder and the decoder.

The encoder-decoder with attention model [4] has been used in machine translation. However, in the relation extraction task, the attention model cannot capture the multi-word entity names. In our preliminary investigation, we found that the attention model yields misalignment between the word and the entity.

The above problem is due to the same words in the names of different entities (e.g., the word University in different university names such as New York University,

`Washington University`, etc.). During training, the model pays more attention to the word `University` to differentiate different types of entities of a similar name, e.g., `New York University`, `New York Times Building`, or `New York Life Building`, but not the same types of entities of different names (e.g., `New York University` and `Washington University`). This may cause errors in entity alignment, especially when predicting the ID of an entity that is not in the training data. Even though we add ⟨`Entity-name, Entity-ID`⟩ pairs as training data (see the Training section), the misalignments still take place.

We address the above problem by proposing an n-gram based attention model. This model computes the attention of all possible n-grams of the sentence input. The attention weights are computed over the n-gram combinations of the word embeddings, and hence the context vector for the decoder is computed as follows.

$$\mathbf{c}_t^d = \left[\mathbf{h}^e; \sum_{n=1}^{|N|} \mathbf{W}^n \left(\sum_{i=1}^{|X^n|} \alpha_i^n \mathbf{x}_i^n \right) \right] \tag{4.8}$$

$$\alpha_i^n = \frac{\exp(\mathbf{h}^{e\top} \mathbf{V}^n \mathbf{x}_i^n)}{\sum_{j=1}^{|X^n|} \exp(\mathbf{h}^{e\top} \mathbf{V}^n \mathbf{x}_j^n)} \tag{4.9}$$

Here, $\mathbf{c}_t^d$ is the context vector of the decoder at time-step t, $\mathbf{h}^e$ is the last hidden state of the encoder, the superscript n indicates the n-gram combination, $\mathbf{x}$ is the word embeddings of input sentence, $|X^n|$ is the total number of n-gram token combination, N indicates the maximum value of n used in the n-gram combinations ($N = 3$ in our experiments), $\mathbf{W}$ and $\mathbf{V}$ are learned parameter matrices, and α is the attention weight. Figure 4.5 illustrates the n-gram attention model.

Training

In the training phase, in addition to the sentence-triple pairs collected using distant supervision (see Section 4.2.2), we also add pairs of ⟨`Entity-name, Entity-ID`⟩ of all entities in the KB to the training data, e.g., ⟨`New York University, Q49210`⟩. This allows the model to learn the mapping between entity names and entity IDs, especially for the unseen entities.

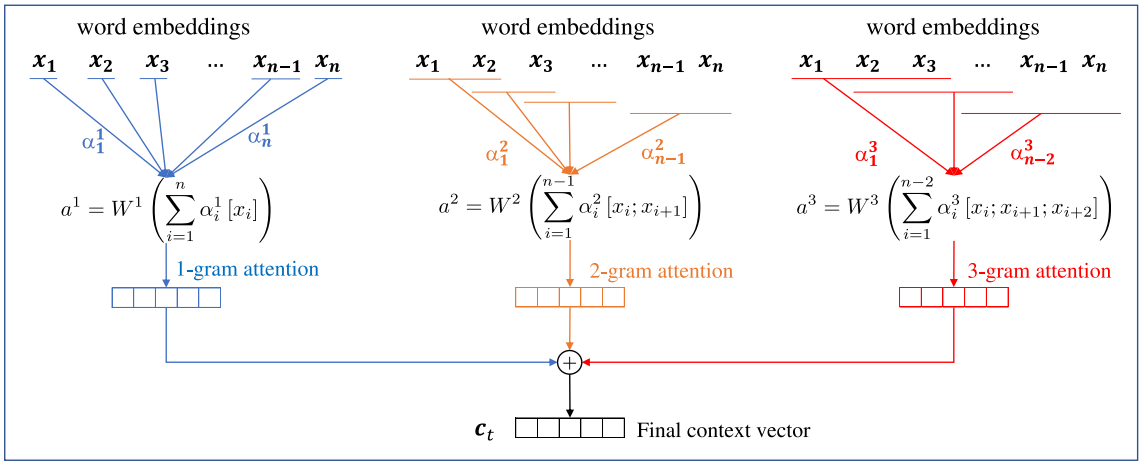


Figure 4.5: N-gram attention model

4.2.5 Triple Generation

The output of the encoder-decoder model is a sequence of the entity and predicate IDs where every three tokens indicate a triple. Therefore, to extract a triple, we simply group every three tokens of the generated output. However, the greedy approach (i.e., picking the entity with the highest probability of the last softmax layer of the decoder) may lead the model to extract incorrect entities due to the similarity between entity embeddings (e.g., the embeddings of `New York City` and `Chicago` may be similar because both are cities in the United States of America). To address this problem, we propose two strategies: re-ranking the predicted entities using a modified beam search and filtering invalid triples using a triple classifier.

The modified beam search re-ranks top-k ($k = 10$ in our experiments) entity IDs that are predicted by the decoder by computing the edit distance between the entity names (obtained from the KB) and every n-gram token of the input sentence. The intuition is that the entity name should be mentioned in the sentence so that the entity with the highest similarity will be chosen as the output.

Our triple classifier is trained with entity embeddings from the joint learning (see Section 4.2.3). Triple classification is one of the metrics to evaluate the quality of entity embeddings [134]. We build a classifier to determine the validity of a triple $\langle s, p, o \rangle$. We train a binary classifier based on the plausibility score $(\mathbf{s} + \mathbf{p} - \mathbf{o})$ (the score to compute

the entity embeddings). We create negative samples by corrupting the valid triples (i.e., replacing the head or tail entity by a random entity). The triple classifier is effective to filter invalid triple such as ⟨New York University, capital of, Manhattan⟩.

There may be multiple unrelated triples in a sentence. Our proposed model handles this problem as follows. First, our encoder captures the latent information (e.g., the entities in the sentence, how many relations between them, etc.) from the input sentence using our proposed n-gram attention model. This latent information is stored in the form of a hidden vector representation of the encoded input sentence. Then, in the triple generation process, this vector representation, along with the decoder's previous hidden state, is used as context vectors for the decoder to guide the generation process. The input vector representation helps the decoder to generate the triples (i.e., predict what the entities and their relationships in a triple are), while the decoder's previous hidden state helps to avoid generating duplicate triples.

It is worth noting that in this paper, our goal is to enrich a KB by adding more relationships between existing entities in a KB (i.e., KB completion). If an input sentence contains unknown entities, our model will generate a special token ⟨UNK⟩. Triples with this special token will be removed from the generated output as we aim to enrich the knowledge base based on the predefined entities and relationships.

4.3 Experiments

We evaluate our model on two real datasets, including WIKI and GEO test datasets (see Section 4.2.2). We use precision, recall, and F1 score as the evaluation metrics.

4.3.1 Hyperparameters

We use grid search to find the best hyper-parameters for the networks. We use 512 hidden units for both the encoder and the decoder. We use 64 dimensions of pre-trained word and entity embeddings (see Section 4.2.3). We use a 0.5 dropout rate for regularization on both the encoder and the decoder. We use Adam [67] with a learning rate of 0.0002.

4.3.2 Baseline Models

We compare our proposed model with three existing models, including **CNN** (the state-of-the-art supervised approach by Lin et at. [80]), **MiniE** (the state-of-the-art unsupervised approach by Gashteovski et al. [47]), **ClausIE** by Corro et al. [27], **CopyR** (a triples generation model based on encoder-decoder framework by [179]), and **CopyR-RL** (a triples generation model based on reinforcement learning by [178]). To map the extracted entities by these models, we use two state-of-the-art NED systems, including **AIDA** [56] and **NeuralEL** [70]. The precision (tested on our test dataset) of AIDA and NeuralEL is 70% and 61%, respectively. To map the extracted predicates (relationships) of the unsupervised approaches output, we use the dictionary-based paraphrase detection. We use the same dictionary that is used to collect the dataset (i.e., the combination of three paraphrase dictionaries, including PATTY [99], POLY [48], and PPDB [44]). We replace the extracted predicate with the correct predicate ID if one of the paraphrases of the correct predicate (i.e., the gold standard) appear in the extracted predicate. Otherwise, we replace the extracted predicate with `"NA"` to indicate an unrecognized predicate. We also compare our **N-gram Attention** model with two encoder-decoder based models, including the **Single Attention** model [4] and **Transformer** model [153].

4.3.3 Results

Table 4.3 shows that the end-to-end models outperform the existing model. In particular, our proposed n-gram attention model achieves the best results in terms of precision, recall, and F1 score. Our proposed model outperforms the best existing model (MinIE) by 33.39% and 34.78% in terms of the F1 score on the WIKI and GEO test dataset, respectively. These results are expected since the existing models are affected by the error propagation of the NED. As expected, the combination of the existing models with AIDA achieves higher F1 scores than the combination with NeuralEL as AIDA achieves higher precision than NeuralEL. CopyR and CopyR-RL achieve low performance as they cannot capture multi-word entity names.

To further show the effect of error propagation, we set up experiments without the

Table 4.3: Performance comparisons of relation extraction models

	Model	WIKI Precision	WIKI Recall	WIKI F1	GEO Precision	GEO Recall	GEO F1
Existing Models	MinIE (+AIDA)	0.3672	0.4856	0.4182	0.3574	0.3901	0.3730
	MinIE (+NeuralEL)	0.3511	0.3967	0.3725	0.3644	0.3811	0.3726
	ClausIE (+AIDA)	0.3617	0.4728	0.4099	0.3531	0.3951	0.3729
	ClausIE (+NeuralEL)	0.3445	0.3786	0.3607	0.3563	0.3791	0.3673
	CNN (+AIDA)	0.4035	0.3503	0.3750	0.3715	0.3165	0.3418
	CNN (+NeuralEL)	0.3689	0.3521	0.3603	0.3781	0.3005	0.3349
	CopyR (+AIDA)	0.3357	0.3001	0.3130	0.3143	0.2932	0.3024
	CopyR (+NeuralEL)	0.3012	0.2971	0.2992	0.3182	0.2711	0.2924
	CopyR-RL (+AIDA)	0.3422	0.3011	0.3232	0.3221	0.3057	0.3154
	CopyR-RL (+NeuralEL)	0.3146	0.3114	0.3125	0.3011	0.2901	0.2921
Encoder-Decoder Models	Single Attention	0.4591	0.3836	0.4180	0.4010	0.3912	0.3960
	Single Attention (+pre-trained)	0.4725	0.4053	0.4363	0.4314	0.4311	0.4312
	Single Attention (+beam)	0.6056	0.5231	0.5613	0.5869	0.4851	0.5312
	Single Attention (+triple classifier)	0.7378	0.5013	0.5970	0.6704	0.5301	0.5921
	Transformer	0.4628	0.3897	0.4231	0.4575	0.4620	0.4597
	Transformer (+pre-trained)	0.4748	0.4091	0.4395	0.4841	0.4831	0.4836
	Transformer (+beam)	0.5829	0.5025	0.5397	0.6181	0.6161	0.6171
	Transformer (+triple classifier)	0.7307	0.4866	0.5842	0.7124	0.5761	0.6370
Proposed	N-gram Attention	0.7014	0.6432	0.6710	0.6029	0.6033	0.6031
	N-gram Attention (+pre-trained)	0.7157	0.6634	0.6886	0.6581	0.6631	0.6606
	N-gram Attention (+beam)	0.7424	**0.6845**	0.7123	0.6816	**0.6861**	0.6838
	N-gram Attention (+triple classifier)	**0.8471**	0.6762	**0.7521**	**0.7705**	0.6771	**0.7208**

canonicalization task (i.e., the objective is predicting a relationship between known entities). We remove the NED pre-processing step by allowing the CNN model to access the correct entities. Meanwhile, we provide the correct entities to the decoder of our proposed model. In this setup, our model achieves 86.34% and 79.11%, while CNN achieves 81.92% and 75.82% in precision over the WIKI and GEO test datasets, respectively.

Our proposed n-gram attention model outperforms the end-to-end models by 15.51% and 8.38% in terms of F1 score on the WIKI and GEO test datasets, respectively. The Transformer model also only yields similar performance to that of the Single Attention model, which is worse than ours. These results indicate that our model captures multi-word entity names (in both datasets, 82.9% of the entities have multi-word entity names) in the input sentence better than the other models.

Table 4.3 also shows that the pre-trained embeddings improve the performance of the model in all measures. Moreover, the pre-trained embeddings help the model to converge faster. In our experiments, the models that use the pre-trained embeddings converge in 20 epochs on average, while the models that do not use the pre-trained embeddings converge in $30 - 40$ epochs. Our triple classifier, combined with the modified beam search, boosts the performance of the model. The modified beam search provides a high recall by extracting the correct entities based on the surface form in the input sentence, while the triple classifier provides a high precision by filtering the invalid triples. We further perform manual error analysis. We found that the incorrect output of our model is caused by the same entity name of two different entities (e.g., the name of `Michael Jordan` that refers to the American basketball player or the English footballer). The modified beam search cannot disambiguate those entities as it only considers the lexical similarity.

4.4 Summary

We proposed an end-to-end relation extraction model for KB enrichment that integrates the extraction and canonicalization tasks. Our model thus reduces the error propagation between relation extraction and NED that existing approaches are prone to. To obtain high-quality training data, we adapt distant supervision and augment it with coreference resolution and paraphrase detection. We propose an n-gram based attention model that better captures the multi-word entity names in a sentence. Moreover, we propose a modified beam search and a triple classification that helps the model to generate high-quality triples.

Experimental results show that our proposed model outperforms the existing models by 33.39% and 34.78% in terms of the F1 score on the WIKI and GEO test dataset, respectively. These results confirm that our model reduces the error propagation between NED and relation extraction. Our proposed n-gram attention model outperforms the other encoder-decoder models by 15.51% and 8.38% in terms of the F1 score on the two real-world datasets. These results confirm that our model better captures the multi-word entity names in a sentence.

Chapter 5

Description Generation for Knowledge Bases Represented as Star-Shaped Graphs

This chapter is based on a paper that has been published in the AAAI Conference on Artificial Intelligence 2020: Sentence Generation for Entity Description with Content-plan Attention. Bayu Distiawan Trisedya, Jianzhong Qi, Rui Zhang.

5.1 Introduction

IN this chapter, we study how to generate an entity description from its properties to enrich a knowledge base. Specifically, the generated descriptions are used to enrich information about entities in a knowledge base, which later can be used in many downstream applications. For example, in question answering systems [161, 167], the generated sentence can be used to describe the entity in the answer. This work complements our efforts in enriching a knowledge base via entity alignment and relation extraction that have been detailed in the previous chapters.

Description generation aims to generate a sentence from a set of properties of a target entity A (i.e., description of the target entity). The properties are in the form of pairs of key and value, i.e., $A = \{\langle k_1; v_1\rangle, \langle k_2; v_2\rangle, ...\langle k_n; v_n\rangle\}$, where k_n is the key of the property and v_n is the value of the property. Table 5.1 illustrates the input and output of the task; in this example, the properties are `name`, `type`, etc. and their values are `"Flinders Street Station"`, `"Station"`, etc. Here, the properties may have been extracted from a table, which makes the task *table-to-text generation*, or from a knowledge graph

Table 5.1: Data-to-text generation example

Input	⟨name; Flinders Street Station⟩ ⟨type; Station⟩ ⟨longitude; 144.9673124⟩ ⟨color; Yellow⟩ ⟨latitude; -37.8182822⟩ ⟨located_in_country; Australia⟩ ⟨located_in_city; Melbourne⟩ ⟨roof; green dome⟩
Output	Flinder Street Station is a railway station with its distinctive yellow facade and green dome in Melbourne, Australia

(KG), which makes the task *triple-to-text generation*. In table-to-text generation, the properties are extracted from a two-column table (e.g., Wikipedia infobox), where the first column indicates the key, and the second column indicates the value of the properties. In triple-to-text generation, the properties are extracted by querying a KG for triples (i.e., ⟨subject,predicate,object⟩) that contain the target entity as the subject. In both cases, the input will form a star-shaped graph with the target entity as the center of the star-shaped graph, the property values as the points of the star, and the property keys as the edges (cf. Figure 5.1).

Recent studies proposed end-to-end models by adapting the encoder-decoder framework. The encoder-decoder framework is a sequence-to-sequence model that has been successfully used in many tasks, including machine translation [21] and data-to-text generation [82]. The adaption of the sequence-to-sequence model for data-to-text generation includes representing the input as a sequence. Hence, the order of properties is essential to guide the decoder to generate a good description [156]. Here, our definition of a proper order of properties is the reasonable order of properties in a well-organized sentence (i.e., a *content-plan*). For example, ⟨name, type, color, roof, located_in_city, located_in_country⟩ is a content-plan for the output sentence in Table 5.1.

Previous studies [6, 81, 82, 128] do not explicitly handle the order of input (i.e., the

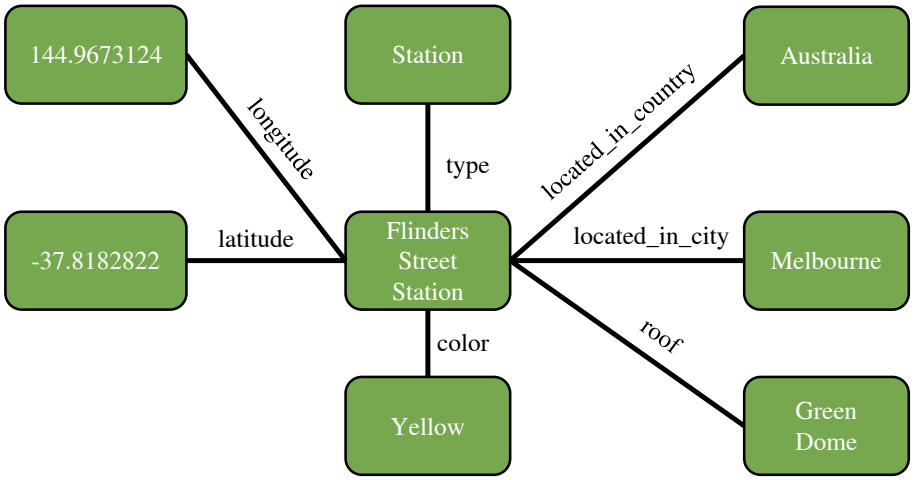


Figure 5.1: Star-shaped graph example

properties). In fact, most data sources do not provide sets of properties with a proper order. For example, the extracted properties as a result of a query from a KG are typically disordered. Meanwhile, the extracted properties from a web table are practically ordered, i.e., the salient properties are relatively ordered but may have noises (non-salient properties) between them, which disrupt the encoding. Moreover, Li et al. [82] reported a decrease in the performance of their model when experimenting on disordered input. Marcheggiani et al. [89] proposed a graph encoder based on Graph Convolution Networks (GCN) [68] to exploit the input structure for generating sentences from a knowledge base. However, GCN fails to capture the relationship between entities of a star-shaped graph since there are no edges between nodes that show the reasonable order of nodes.

Puduppully et al. [112] proposed Neural Content Planning (NCP), which is a two-stage model that includes content-planning to handle disordered input. First, NCP uses pointer networks (i.e., content-planner) [157] to generate a content-plan. Then, the generated content-plan is used as the input of the encoder-decoder model (i.e., text generator) [4] to generate a description. However, this two-stage model suffers from error propagation between the content-planner and the text generator. The generated content-plan may contain errors (e.g., missing one or more properties that should be mentioned in the description) that lead the text generator to produce an incomplete description.

In this chapter, we address the issues above by proposing an end-to-end model that jointly learns the content-planner and the text generator by integrating the content-plan in the attention model of an encoder-decoder model. The challenge of the integration is to align the learned content-plan and the generated description. To address this problem, we propose the *content-plan-based bag of tokens* attention model by adapting the coverage mechanism [150] to track the order of properties in a content-plan for computing the attention of the properties. This mechanism helps the attention module of the encoder-decoder model captures the most salient properties at each time-step of the description generation phase in a proper order. Unlike the existing data-to-text generation models which treat the input as a sequence of properties, our model treats the input as a bag of tokens and uses pointer networks to learn a content plan to handle disordered properties. Our model maintains the original data (i.e., the original set of properties) as the input of the text generator while exploiting the learned content-plan to highlight the properties to be mentioned. Hence, our model is able to reduce the error propagation between the content-planner and the text generator. To collect training data for the content-planner, we use string matching for extracting the order of properties mentioned in the description as the target content-plan.

Our contributions are summarized as follows:

- We propose an end-to-end model that employs joint learning of content-planning and text generation to handle disordered input for generating a description of an entity from its properties. The model reduces error propagation between the content-planner and the text generator, which two-stage models are prone to.
- We propose a *content-plan-based bag of tokens* attention model to effectively capture salient properties in a proper order based on a content-plan.
- We evaluate the proposed model over two real-world datasets. The experimental results show that our model consistently outperforms state-of-the-art baselines for data-to-text generation [82, 112].

The rest of this chapter is organized as follows. Section 5.2 defines the studied problem. Section 5.3 details the proposed model. Section 5.4 presents the experimental results.

Section 5.5 summarizes the chapter.

5.2 Preliminary

We start with the problem definition. Let A be a set of properties of an entity in the form of pairs of key and value in any order, i.e., $A = \{\langle k_1; v_1\rangle, \langle k_2; v_2\rangle, \langle k_3; v_3\rangle, ...\langle k_n; v_n\rangle\}$, where k_n is the key of the property and v_n is the value of the property. We consider A as the input and aim to generate a sentence $S = \langle t_1, t_2, t_3, ..., t_l\rangle$ as the description of an entity, where t_l is a token at position l in the sentence. Table 5.1 illustrates the input and output of the task.

Most data-to-text generation models are built on top of an encoder-decoder framework [82, 93, 128, 166]. We first discuss the encoder-decoder framework [4] and its limitation when generating text from disordered input.

5.2.1 Encoder-Decoder Framework

The encoder-decoder framework is a sequence-to-sequence learning model that takes a variable-length input T and generates a variable-length output T' where the length of T and T' may differ. The encoder reads each token of the input sequentially and computes a hidden state of each token. The hidden state of the last token represents a summary of the input sequence in the form of a fixed-length vector representation (i.e., context vector **c**). The decoder is trained to generate a sequence by predicting the next token given the previous hidden state of the decoder and the context vector **c**. This framework has been successfully applied in machine translation [21] to translate a sequence of words from one language to another.

In data-to-text generation, encoder-decoder is used to generate text (e.g., entity description) from structured data (e.g., a set of properties of an entity). Here, the encoder learns to encode the properties into a fixed-length vector representation, which is used as a context vector by the decoder to generate a description. Different from machine translation, in data-to-text generation, the input (i.e., properties) may be disordered, and linearizing the input may not yield the proper order of the properties. Hence, reading

the disordered input sequentially may produce an improper context vector.

Bahdanau et al. [4] proposed an attention model that improves the performance of sequence-to-sequence models. The attention model allows the encoder to dynamically compute the context vector for each decoding time-step by computing the weighted sum of each hidden state of the encoder. The weight represents the importance score of each token of the properties and helps the encoder computes a specific context vector for each decoding time-step. However, the computations of the context vector are based on the hidden states of the encoder, which may not be appropriate for disordered input because a hidden state represents a summary of the previous tokens that do not hold the proper order. Besides, if we use the embeddings of the input (i.e., embeddings of the property token) instead of the hidden states to compute the attention, the model may not capture the relationships between properties.

Next, we detail our model to address these limitations.

5.3 Proposed Model

We present an overview of our proposed model in Section 5.3.1. We detail the components of the proposed model afterwards, including the dataset collection module in Section 5.3.2, the content-plan generation module in Section 5.3.3, and the description generation module in Section 5.3.4.

5.3.1 Solution Framework

Figure 5.2 illustrates the overall solution framework. Our framework consists of three components: a *data collection module*, a *content-plan generation module*, and a *description generation module*.

In the data collection module (cf. Section 5.3.2), we collect a dataset in the form of triples of properties, content-plan, and entity description. The properties are extracted by querying Wikidata for triples that contain the target entity as the subject. The description is obtained from Wikipedia by extracting the first sentence of the Wikipedia page of the target entity. The content-plan is extracted by finding the order of properties in the

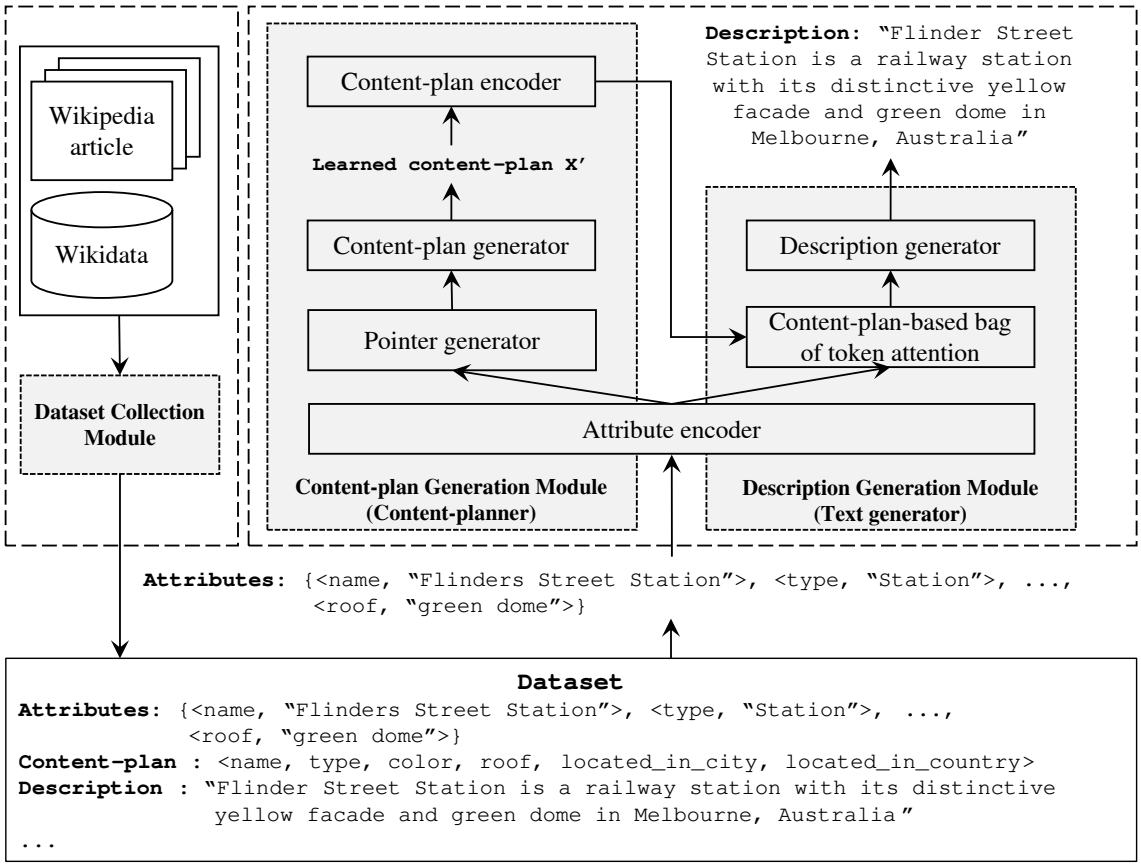


Figure 5.2: Overview of our proposed solution for entity description generation

description using string matching.

In the content-plan generation module (content-planner, cf. Section 5.3.3), we train pointer networks [157] to learn a content-plan that helps the attention model of the description generation module highlights the properties in a proper order. This module consists of four components: (1) a *property encoder* that encodes a set of properties into a vector by computing the average of the linear transformation of each token embeddings of the properties; (2) a *pointer generator* that generates a sequence of indexes (pointers) that represents the order of properties in the description; (3) a *content-plan generator* that generates the content-plan based on the learned pointers; and (4) a *content-plan encoder* that encodes the learned content-plan to be used in the description generation module.

In the description generation module (text generator, cf. Section 5.3.4), we integrate the content-plan into the attention mechanism of the encoder-decoder model [4]. We

use the same encoder as in the content-plan generation module that treats the input (i.e., properties) as a bag of tokens to ensure that the same set of properties with different orders have the same representation. We do not use the recurrent model (e.g., LSTM [55], GRU [21]) to encode the properties because they may compute improper context from disordered input (cf. Section 5.2.1). However, we use LSTM (i.e., content-plan encoder) to encode the learned content-plan that holds the proper order of properties to capture the relationships between properties. To integrate the learned content-plan into the attention mechanism of the encoder-decoder model, we propose the *content-plan-based bag of tokens* attention model by adapting the coverage mechanism [150] to track the order of properties in a content-plan for computing the attention of the properties. This way, our proposed attention model selects the salient properties conditioned by the content-plan and hence provides a better context (i.e., attention of the properties in an ordered fashion) for each decoding time-step.

5.3.2 Dataset Collection

We aim to generate a description of an entity from its properties where the properties may be disordered. To handle disordered input, we propose a model that performs joint learning of content-planning and text generation. To train such a model, we need a large volume of fully labeled training data in the form of triples of properties, content-plan, and entity description.

Following Lebret et al. [72], we extract the first sentence of a Wikipedia page of a target entity as the description. Different from Lebret et al. [72], who collected a specific type of entities (e.g., `Person`), we do not restrict the type of entities to be collected. We extract the properties of a target entity by querying Wikidata for triples that contain the target entity as the subject. In other words, we extract the direct relationships (i.e., properties) of an entity which form a star-shaped graph. We query the properties from Wikidata instead of extracting from Wikipedia infobox to avoid additional processing (e.g., HTML tag removal, normalization, etc.).

We use string matching to find the order of properties that are mentioned in the description as the content-plan. First, for each matched property, we store their position

(index of the first character of the mentioned property value) in the description. Then, we sort the matched properties based on their position ascendingly. For the example in Table 5.1, the extracted content plan is ⟨`name, type, color, roof, located_in_city, located_in_country`⟩.

Our proposed model is trained to generate a description from a set of properties of a target entity, which includes selecting salient properties to be described. Since we automatically extract the description from Wikipedia, the extracted description may contain information (i.e., entities) that are not listed in the related extracted properties, which creates noises in the dataset. This problem may be caused by the delayed synchronization of a KG (i.e., the Wikipedia page has been updated, but the Wikidata records have not been updated yet). The synchronization problem often occurs on frequently updated information such as the current club of a football player, the latest movie of an actor, etc. Besides the synchronization problem, the string-matching errors used in the algorithm may also create noises in the training data. Hence, to obtain high-quality data, we filter descriptions that contain noises. First, we use a Named Entity Recognizer[1] to detect all entities in a description. Then, we remove descriptions that contain any entity that is not listed in the related extracted properties.

The collected dataset contains 152,231 triples of properties, content-plan, and description (we call it the **WIKIALL** dataset). The dataset contains 53 entity types with an average of 15 properties per entity and an average of 20 tokens per description. For benchmarking, we use the **WIKIBIO** dataset [72], which contains 728,321 biographies from Wikipedia (i.e., this dataset only contains one entity type: `PERSON`). The average number of properties per entity of the WIKIBIO dataset is 19, and the average number of tokens per description is 26. We split each dataset into a train set (80%), a dev set (10%), and a test set (10%).

5.3.3 Content-plan Generation

We adapt pointer networks [157] to learn a content-plan given a set of properties (i.e., we use the pairs of properties and content-plan from the dataset to train the networks). The

[1] https://spacy.io

Table 5.2: Input representation of the proposed description generation model

Key	Value	Forward position	Reverse position
name (k_1^1)	Flinders (v_1^1)	1 (f_1^1)	3 (r_1^1)
name (k_1^2)	Street (v_1^2)	2 (f_1^2)	2 (r_1^2)
name (k_1^3)	Station (v_1^3)	3 (f_1^3)	1 (r_1^3)
type (k_2^1)	Station (v_2^1)	1 (f_2^1)	3 (r_2^1)
longitude (k_3^1)	144.9673124 (v_3^1)	2 (f_3^1)	2 (r_3^1)
...	...	...	...
roof (k_n^1)	green (v_n^1)	1 (f_n^1)	2 (r_n^1)
roof (k_n^2)	dome (v_v^2)	2 (f_n^2)	1 (r_n^2)

pointer networks model uses an attention mechanism to generate a sequence of pointers that refer to the input so that it is suitable to rearrange the properties as a content-plan. This module consists of four components, including a property encoder, a pointer generator, a content-plan generator, and a content-plan encoder.

Property Encoder

The property encoder takes a set of properties $A = \{\langle k_1; v_1\rangle, \langle k_2; v_2\rangle, ...\langle k_n; v_n\rangle\}$ as the input. Here, the value of a property may consist of multiple tokens (i.e., $v_n = \langle v_n^1, v_n^2, ...v_n^j\rangle$). We transform the multiple tokens into a single token representation and add positional encoding to maintain its internal order. Thus, the properties can be represented as $A = [\langle k_1^1, v_1^1, f_1^1, r_1^1\rangle, \langle k_1^2, v_1^2, f_1^2, r_1^2\rangle, ..., \langle k_n^j, v_n^j, f_n^j, r_n^j\rangle]$ where f_n^j and r_n^j are the forward and reverse positions, respectively (cf. Table 5.2). We call the quadruple of key, value, forward position, and reverse position as *property-token*. The representation of each property-token $\mathbf{x_a}$ is computed as follows.

$$\mathbf{z_k}^j{}_n = \tanh(\mathbf{W_k}[\mathbf{k}_n^j; \mathbf{f}_n^j; \mathbf{r}_n^j] + \mathbf{b_k}) \tag{5.1}$$

$$\mathbf{z_v}^j{}_n = \tanh(\mathbf{W_v}[\mathbf{v}_n^j; \mathbf{f}_n^j; \mathbf{r}_n^j] + \mathbf{b_v}) \tag{5.2}$$

$$\mathbf{x_a}^j{}_n = \tanh(\mathbf{z_k}^j{}_n + \mathbf{z_v}^j{}_n) \tag{5.3}$$

where $[;]$ indicates vector concatenation, $\mathbf{b}$ indicates bias vector, and $\mathbf{W_k}$ and $\mathbf{W_v}$ are learned parameters. $\mathbf{z_k}$ and $\mathbf{z_v}$ are the vector representations of the properties' key and value, respectively. To ensure that the same set of properties with different orders have the same representation, we use the element-wise average of property-token vectors as the vector representation of a set of properties. This vector is used as the initial hidden state of the decoder of the pointer networks (i.e., the pointer generator) and the encoder-decoder model (i.e., the text generator) since the property encoder is shared with the description generation module (cf. Section 5.3.4).

Pointer Generator

Given a sequence of property-token vectors that are computed by the property encoder $X = \langle \mathbf{x_a}1, ..., \mathbf{x_a}m \rangle$ (m is the number of property-tokens in the input), the pointer generator aims to generate a sequence of pointer-indexes $I = \langle i_1, ..., i_g \rangle$ (g is the number of properties-tokens in the target content-plan). Here, i_g indicates an index that points to a property-token. The pointer generator uses LSTM to encode the property-token that are selected as part of the content-plan in the previous time-step $\mathbf{x}_{i_{g-1}}$ as a context vector (cf. Eq. (5.4)) to compute the attention of the properties. The pointer generator predicts the next index by selecting property-token with the highest attention (cf. Eq. (5.6)) that are computed as follows.

$$\mathbf{c_{ptr}}_g = f_{lstm}(\mathbf{x}_{i_{g-1}}) \tag{5.4}$$

$$\mathbf{u_{ptr}}_g = \tanh(\mathbf{W_{p}}_1 \mathbf{x_a} + \mathbf{W_{p}}_2 \mathbf{c_{ptr}}_g) \tag{5.5}$$

$$\hat{i}_g = \text{softmax}(\mathbf{u_{ptr}}_g) \tag{5.6}$$

Here, $\hat{i}_g$ is the pointer-index output probability distribution over the vocabulary (in this case, the vocabulary is the property-token input), f_{lstm} is a single LSTM unit, and $\mathbf{W_{p}}_1$ and $\mathbf{W_{p}}_2$ are learned parameters. The pointer generator is trained to maximize the conditional log-likelihood:

$$p(I_d \mid A_d) = \sum_g \sum_{j=1}^{j=m} {i'}_{g,j} \times \log \hat{i}_{g,j} \tag{5.7}$$

$$J_{ptr} = \frac{1}{D} \sum_{d=1}^{D} -\log p(I_d \mid A_d) \tag{5.8}$$

where (A_d, I_d) is a pair of properties and target pointer-index (generated by finding the position of the property-token of the target content-plan in the original input) given for training, i' is the matrix of the target pointer-index over the vocabulary, D is the number of records in the dataset and J_{ptr} is the objective function of the pointer generator.

Content-plan Generator and Encoder

The pointer-index I is a sequence of indexes that refers to the property-token X in a proper order that represents a content-plan. Hence, the content-plan generator uses the pointer-index to rearrange the sequence of property-token into a content-plan X'. In the content-plan encoder, we use LSTM to encode the learned content-plan X' to capture the relationships between properties. The hidden states of the content-plan encoder $\langle \mathbf{x_{cp}}_1, ..., \mathbf{x_{cp}}_g \rangle$ are forwarded to the text generator to help its attention model select properties in a proper order.

5.3.4 Description Generation

We adapt the encoder-decoder model [4] to generate entity description by integrating a content-plan to help the attention mechanism of the encoder-decoder computes the properties to be mentioned and their order. For this adaptation, we propose the *content-plan-based bag of tokens* attention model. This model works as follows.

Content-plan-based Bag of Tokens Attention

We use the same encoder as in the content-plan generation module (cf. Section 5.3.3). This encoder treats the properties as a bag of tokens to allow our model to handle disordered input. However, this representation does not capture the relationships between properties. To capture the relationships between properties, we use LSTM to encode the content-plan (cf. Section 5.3.3). We integrate the encoded content-plan into the encoder-decoder model to help the attention mechanism of the model selects the properties in a proper order. This way, our proposed model has two advantages. First, our model yields the same vector representation for the same set of properties regardless of their order

while capturing the relationships between properties via the content-plan. Second, our model computes a context vector based on the original input (i.e., properties) and the content-plan, and hence reduces the error propagation (e.g., missing property errors).

The integration of the learned content-plan in the encoder-decoder model is done by adapting the coverage mechanism [126, 150] as follows. First, we use a coverage vector $\mathbf{d_{cov}}$ to keep track of the content-plan history. We use the sum of a content-plan attention distribution $\mathbf{a_{cp}}_{l'}$ from the previous decoding (i.e., description generator) time-step to maintain the information about which properties in the content-plan have been exposed (cf. Eq. (5.9))). Second, we use the coverage vector $\mathbf{d_{cov}}$ to compute the attention (weight) of the content-plan $\mathbf{a_{cp}}$ (cf. Eq. (5.10) and (5.11)). Then, the content-plan attention is used as an additional input to compute the attention of the property-token $\mathbf{a_d}$ (cf. Eq. (5.12)). Finally, the property-token attention is used to compute a context vector $\mathbf{c_d}$ for the decoder (cf. Eq. (5.13)).

$$\mathbf{d_{cov}}_l = \sum_{l'=0}^{l-1} \mathbf{a_{cp}}_{l'} \tag{5.9}$$

$$\mathbf{u_{cp}}_l = \tanh(\mathbf{W_{c1}}\mathbf{x_{cp}} + \mathbf{W_{c2}}\mathbf{h_d}_{l-1} + \mathbf{w_{c3}}\mathbf{d_{cov}}_l) \tag{5.10}$$

$$\mathbf{a_{cp}}_l = \sum_g \text{softmax}(\mathbf{u_{cp}}_l)\mathbf{x_{cp}}_g \tag{5.11}$$

$$\mathbf{a_d}_l = \tanh(\mathbf{W_{d1}}\mathbf{x_a} + \mathbf{W_{d2}}\mathbf{h_d}_{l-1} + \mathbf{W_{d3}}\mathbf{a_{cp}}_l) \tag{5.12}$$

$$\mathbf{c_d}_l = \sum_m \text{softmax}(\mathbf{a_d}_l)\mathbf{x_a}_m \tag{5.13}$$

Here, $\mathbf{h_d}_l$ is the decoder hidden state at time-step l, and $\mathbf{W}$ and $\mathbf{w}$ are learned parameters.

Description Generator

We use LSTM for the description generator (i.e., the decoder). The decoder predicts the next token of the description conditioned by the previous hidden state of the decoder $\mathbf{h_d}_{l-1}$, the previous generated token t_{l-1}, and the context vector $\mathbf{c_d}_l$.

$$\mathbf{h_d}_l = f_{lstm}([\mathbf{h_d}_{l-1}; \mathbf{t}_{l-1}; \mathbf{c_d}_l]) \tag{5.14}$$

$$\hat{t}_l = \text{softmax}(\mathbf{V}\mathbf{h_d}_l) \tag{5.15}$$

Here, $\hat{t}_l$ is the output probability distribution over the vocabulary, and $\mathbf{V}$ is the hidden-to-output weight matrix. The decoder is trained to maximize the conditional log-likelihood:

$$p(S_d \mid A_d) = \sum_{l} \sum_{j=1}^{j=|V|} t'_{l,j} \times \log \hat{t}_{l,j} \tag{5.16}$$

$$J_{dec} = \frac{1}{D} \sum_{d=1}^{D} -\log p(S_d \mid A_d) \tag{5.17}$$

$$J = J_{ptr} + J_{dec} \tag{5.18}$$

where (A_d, S_d) is a pair of properties and entity description given for training, t' is the matrix of the target token description over the vocabulary V, J_{dec} is the objective function of the description generator, and J is the overall objective function of our proposed model.

5.4 Experiments

We evaluate our model on two real-world datasets, including WIKIALL and WIKIBIO datasets. The properties in WIKIALL are disordered since they are the result of a query to Wikidata. Meanwhile, the properties in WIKIBIO are practically ordered, i.e., the salient properties are relatively ordered but may have noises (non-salient properties) between them. To test on disordered properties of the WIKIBIO dataset, we randomly shuffle the properties. We use BLEU, ROUGE, METEOR, and TER as the evaluation metrics.

5.4.1 Hyperparameters

We use 512 hidden units for both the pointer networks (content-planner) and the encoder-decoder (text generator). We use 128, 64, and 8 dimensions of word embeddings (property value token), type embeddings (property key), and position embeddings, respectively. We use a 0.5 dropout rate for regularization. We use Adam [67] with a learning rate of 0.0001 for optimization.

Table 5.3: Performance comparisons of description generation models

(a) Results on WIKIBIO dataset

Model	WIKIBIO (disordered)				WIKIBIO (ordered)			
	BLEU	ROUGE	METEOR	TER	BLEU	ROUGE	METEOR	TER
MED	40.12	36.02	31.70	55.60	42.54	38.14	32.80	54.20
GTRLSTM	42.64	38.35	32.60	54.40	42.06	37.90	32.20	54.80
NCP	43.07	38.76	33.90	53.20	43.12	38.82	33.90	53.30
FGDA	42.31	38.93	32.20	55.00	44.59	40.20	34.10	52.10
Proposed	**45.32**	**40.80**	**34.40**	**51.50**	**45.46**	**40.31**	**34.70**	**51.30**

(b) Results on WIKIALL dataset

Model	WIKIALL			
	BLEU	ROUGE	METEOR	TER
MED	61.77	56.91	44.20	30.70
GTRLSTM	62.71	57.02	44.30	29.50
NCP	63.21	57.28	44.30	28.50
FGDA	62.61	57.11	44.10	30.20
Proposed	**65.12**	**58.07**	**45.90**	**27.50**

5.4.2 Baseline Models

We compare our proposed model with four existing models including (1) the Field Gating with Dual Attention model (**FGDA**), which is the state-of-the-art table-to-text generation model [82]; (2) the Graph-based Triple LSTM encoder model (**GTRLSTM**) that uses topological sort and breadth-first traversal to select the entity order in the encoding process, which is our initial work on triple-to-text generation model described (to be described) in Chapter 6; (3) the Neural Content-Planning model (**NCP**), which is a data-to-text generation model that uses content-plan as one of its features [112]; and (4) the modified encoder-decoder model (**MED**), which is a modification of the standard encoder-decoder model that uses the embeddings of its input instead of the hidden state of the encoder to compute the attention.

5.4.3 Results

Table 5.3 shows that our proposed model achieves a consistent improvement over the baselines, and the improvement is statistically significant, with $p < 0.01$ based on the t-test of the BLEU scores. We use MultEval to compute the p-value based on approximate randomization [24]. Our model achieves higher BLEU and ROUGE scores than the baselines, which indicate that our model generates descriptions with a better order of property mention. Moreover, the better (lower) TER scores indicate that our model generates a concise description (i.e., following the content-plan).

On disordered input experiments, the content-plan based models (i.e., our proposed model and NCP) achieve stable performance with our model getting the highest score on all metrics. These results show that content-planning helps neural data-to-text generation models select and arrange the data (i.e., properties) to be mentioned in the text.

The content-planner (the pointer networks) achieves 83.41 and 87.12 BLEU scores on the WIKIBIO and WIKIALL datasets, respectively. We further conduct experiments to show that our model reduces error propagation between the content-planner and the text generator. We use the content-plan gold standard (i.e., the target content-plan extracted from the description, cf. Section 5.3.2) as the input of the text generator. On this setup, our model achieves comparable performance with NCP. Our model achieves 46.5 and 66.97 BLUE scores on the WIKIBIO and WIKIALL datasets, respectively. Meanwhile, NCP achieves 46.3 and 66.69 BLUE scores on the WIKIBIO and WIKIALL datasets, respectively. These results are expected because both models take the same content-plan.

Human Evaluations

We conduct manual evaluations on the generated descriptions using three metrics, including *correctness*, *grammaticality*, and *fluency*. *Correctness* is used to measure the semantics of the generated description (i.e., contains wrong order of property mention or not, e.g., "`born in USA, New York`"); *grammaticality* is used to rate the grammatical and spelling errors; and *fluency* is used to measure the fluency of the output (e.g., contain repetition or not). For each metric, a score of 3 is given to output that contains no errors;

Table 5.4: Human evaluation results

Model	Correctness	Grammaticality	Fluency
MED	2.25	2.32	2.26
GTRLSTM	2.31	2.40	2.36
NCP	2.54	2.68	2.51
FGDA	2.51	2.58	2.54
Proposed	**2.68**	**2.76**	**2.57**

a score of 2 is given to output that contains one error; a score of 1 is given to output that contains more than one error. We randomly choose 300 records of the WIKIALL dataset along with the output of each model. We manage to get six annotators who have studied English for at least ten years and completed education in an English environment for at least two years. The total time spent for these evaluations is around 250 hours. Table 5.4 shows the results of the human evaluations. The inter-annotator agreement measured by Fleiss' kappa [41] is 0.47, which indicates moderate agreement. The results confirm the automatic evaluations in which our proposed model achieves the best scores.

Discussion

Our model is a statistical model that performs joint learning of content-planning and text generation. Hence, it needs a large training set in the form of triples of properties, content-plan, and description. However, extracting a content-plan from a description is a non-trivial task. We use string matching to find the order of properties in the description as a content-plan. However, string matching does not capture the semantic similarity between properties and text. For example, string matching cannot capture the similarity between `United States` and `American`, even though the word `American` is commonly used to describe a United States citizen.

5.5 Summary

In this chapter, we studied the problem of description generation for enriching a knowledge base. We proposed an end-to-end data-to-text generation model on top of an encoder-

decoder framework that includes content-planning to address the problem of disordered input. Our model employs joint learning of content-planning and text generation to reduce error propagation between them for generating a description of an entity from its properties. To integrate a content-plan into the encoder-decoder framework, we propose the *content-plan-based bag of tokens* attention model. Our attention model effectively captures salient properties in a proper order. Experimental results show that our proposed model outperforms the baselines and achieves the highest score in all metrics on the WIKIALL and WIKIBIO test datasets. Moreover, our model achieves 45.46 and 45.32 in terms of BLEU score on the ordered and disordered WIKIBIO test dataset. These results show that our proposed model obtains stable performance on disordered input and achieves a consistent improvement over the baselines by up to 5%.

Chapter 6

Description Generation for Knowledge Bases Represented as Arbitrary-Shaped Graphs

This chapter is based on a paper that has been published in the Annual Meeting of the Association for Computational Linguistics (ACL) 2018: GTR-LSTM: A Triple Encoder for Sentence Generation from RDF Data. Bayu Distiawan Trisedya, Jianzhong Qi, Rui Zhang, Wei Wang.

6.1 Introduction

IN Chapter 5, we study *text generation* [116] from structured data. Specifically, we aim to translate triples into a natural sentence that describes an entity in a KG. We call this task *triple-to-text generation*. However, in the previous chapter, we only consider a set of triples in the form of a star-shaped graph, where there is a central entity that appears in every triple of interest. As illustrated in Fig. 6.1(a), The entity `Flinders Street Station` is the central entity, and the other entities (or literals) connect to the central entity directly. The star-shaped graph is easy to extract since many KBs allows unnormalized forms, but it is less natural in representing the real-world relationships between entities. For example, the relationship between `Australia` and `Melbourne` may not be properly defined. In contrast, the graph in Fig. 6.1(b) represents the relationship between entities in a more natural way. However, it may form an arbitrary-shaped graph as opposed to a star-shaped graph, which is more challenging for a machine to process.

In this chapter, we study triple-to-text generation from an arbitrary-shaped graph to generate a description of entities in a knowledge graph. Given a set of triples re-

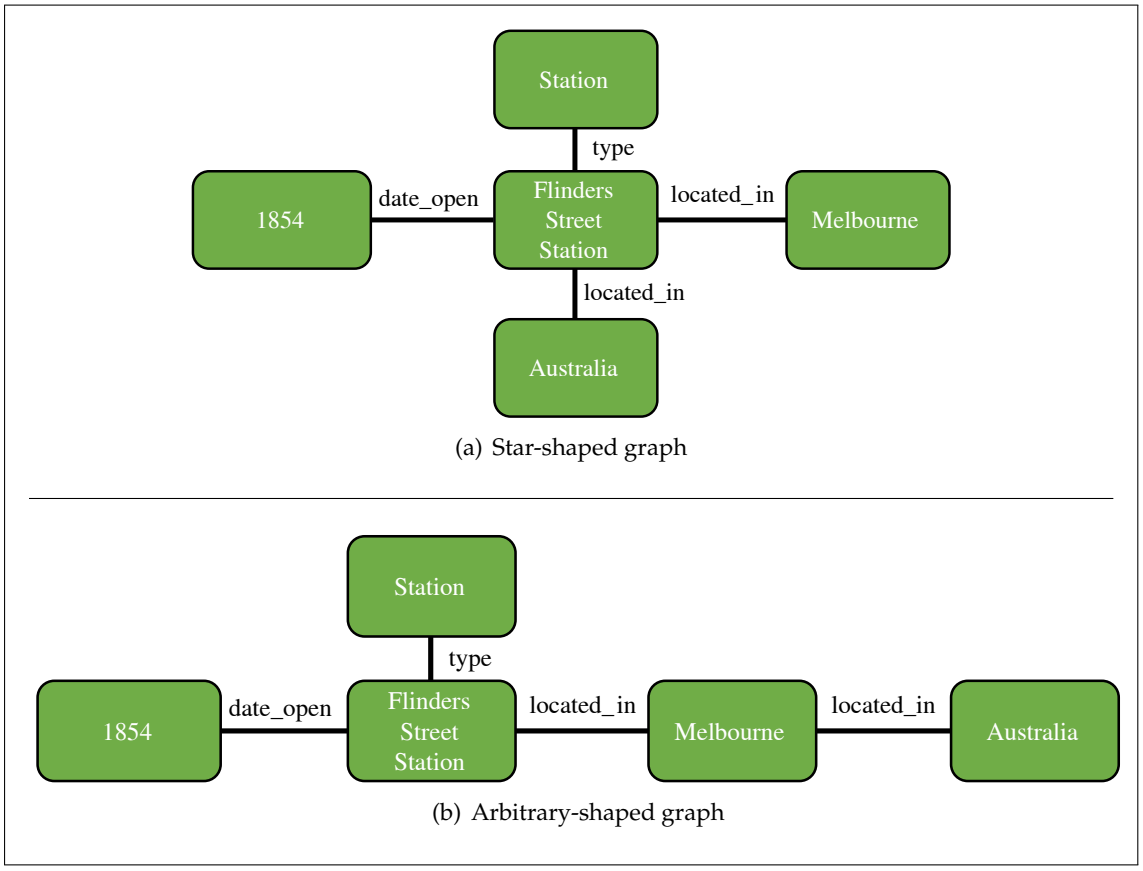


Figure 6.1: Different shapes of graphs

lated to a target entity [⟨John Doe, birth place, London⟩, ⟨John Doe, birth date, 1967-01-10⟩, and ⟨London ,capital of, England⟩], we aim to generate a natural sentence that describes the target entity. The description incorporates the information of the triples and is easier to be understood by humans. In this example, the generated sentence is "John Doe was born on January 10, 1967 in London, the capital of England". Table 6.1 illustrates such an example. The generated description can be used to enrich a knowledge graph, which later can be used in many downstream applications. For example, in question answering [161, 167], the generated sentence can be used to describe the entity in the answer.

Traditional triple-to-text generation approaches use domain-specific rules. Bontcheva and Wilks [10] created rules to generate sentences in the medical domain, while Cimiano

Table 6.1: Data-to-text generation from an arbitrary-shaped graph

Triples	⟨John Doe, birth place, London⟩ ⟨John Doe, birth date, 1967-01-10⟩ ⟨London, capital of, England⟩
Target Sentence	John Doe was born on January 10, 1967 in London, the capital of England.

et al. [23] created rules to generate step by step cooking instructions. Rule-based approaches need a lot of human efforts to create the rules, which mostly cannot deal with complex or novel cases. Recent studies in text generation propose neural generation models. Lebret et al. [72] developed a conditional neural language model to generate the first sentence of a biography. Another approaches in biography summarization use encoder-decoder frameworks [6, 82, 128]. Mei et al. [93] proposed an encoder-aligner-decoder architecture to generate weather forecasts. These models do not need predefined rules and hence generalize better to open domain data.

A straightforward adaptation of neural text generation models for triple-to-text generation is to use the encoder-decoder model [4, 21] by first concatenating the elements of the triples into a linear sequence and then feeding the sequence as the model input to learn the corresponding target sentence. We implemented such a model (detailed in Section 6.2.4) that ranked top in the WebNLG Challenge 2017 [46]. This Challenge has a primary objective of generating syntactically correct natural sentences from a set of triples. Our model achieves the highest global scores on the automatic evaluation, outperforming competitors that use rule-based methods, statistical machine translation, and neural machine translation [46].

Simply concatenating the elements in the triples may lose the relationship between entities that affects the semantics of the resulting sentence (cf. Table 6.3). Recent studies in triple-to-text generation proposed a graph encoder to exploit the input structure. Vougioklis et al. [159] proposed a triple encoder using feed-forward neural networks. This encoder is good at capturing the relationships between entities in a triple (*intra-triple relationships*). Still, it may fail to capture the relationships between entities in related triples

(*inter-triple relationships*). Marcheggiani and Perez-Beltrachini [89] employed graph convolutional networks (GCN) [68] as the encoder of the input. GCN is good at capturing the relationship between an entity and its immediate neighbors but may fail to capture long-range relationships between entities.

To address the limitations of the existing models, we propose a novel graph-based triple encoder model named *GTR-LSTM* that maintains the structure of the triples as a small knowledge graph. This model computes the hidden state of each entity in a graph that preserves the intra-triple and inter-triple relationships, which helps to achieve more accurate sentences. Capturing both types of relationships in a knowledge graph leads to two problems: (1) how to deal with a cycle in a knowledge graph; (2) how to deal with multiple relationships between two entities in a knowledge graph (e.g., `John Doe` and `London` may have multiple relationships `birth place` and `live in`).

We handle the aforementioned difficulties as follows. We devise a supervised topological traversal algorithm to encode a graph in more natural order by taking supervision signals on the processing order from *entity-order aware embeddings*. Here, the entity-order refers to the order of entities occurrences in natural sentences. To learn such entity-order aware embeddings, we train a translation-based graph embedding model over a word-entity graph. The word-entity graph is a KG containing triples from which sentences are to be generated, combined with *entity-word co-occurrence triples* and *entity-order triples* extracted from a text corpus. Here, an entity-word co-occurrence triple is formed by word-pairs that co-occur within a window in a sentence. An entity-order triple represents a *previous relationship* between two entities. To handle multiple relationships between entities, we propose a novel GTR-LSTM unit that aggregates both an entity and its relationships, as opposed to the standard LSTM unit [55] that can only take one input.

The proposed model differs from existing non-linear LSTM models, such as Tree LSTM [147] and Graph LSTM [76], in addressing the aforementioned problems. In particular, Tree LSTM does not allow cycles, while the proposed model handles cycles to encode the input graph by first traversing the input graph as described above, and then using an attention model to capture the global information of the knowledge graph. Meanwhile, Graph LSTM only allows a single relationship between entities, while the

proposed model handles multiple relationships by modifying the input, forget, and output gates of the LSTM unit to aggregate an entity and its relationships.

To further enhance the capability of our model to handle unseen entities, we use entity masking, which maps entities in the model training pairs to their types, e.g., we map an entity (literal) `1967-01-10` to a type symbol `DATE` in the training pairs. This way, our model can learn to handle any date entities rather than just `1967-01-10`. This method is particularly helpful when the training data is limited.

Our contributions are summarized as follows:

- We propose a graph-based triple encoder to optimize the amount of information preserved in the input of the model. The proposed model can handle cycles to capture the intra-triple and inter-triple relationships between entities in a KG. The proposed model also handles multiple relationships between entities.

- To handle multiple relationships between entities, we propose a GTR-LSTM unit that aggregates both an entity and its relationships in a single unit.

- To handle cycles, we devise a supervised topological traversal that takes supervision signals based on the order of entities in natural sentences, which makes the encoding more robust.

- To capture the order of entities in natural sentences, we present an entity-order aware translation-based graph embedding model that is trained over a word-entity graph. The learned embeddings maintain the mentioning order of entities in sentences, which helps the encoder to encode the input graph in a more natural order.

- We evaluate the proposed framework and model over two real datasets. The results show that our model outperforms the state-of-the-art neural triple-to-text generation models [89, 159] consistently.

The rest of this chapter is organized as follows. Section 6.2 details the proposed model. Section 6.3 presents the experimental results. Section 6.4 summarizes the chapter.

6.2 Proposed Model

We start with the problem definition. We consider a set of triples as the input, which is denoted by $T = \{t_1, t_2, ..., t_n\}$ where a triple t_n consists of three elements (subject s_n, predicate p_n, and object o_n), $t_n = \langle s_n, p_n, o_n \rangle$. Every element of the triple can contain multiple words. We aim to generate a sentence that consists of a sequence of words $Y = \langle y_1, y_2, ..., y_k \rangle$, such that the relationships in the input triples are correctly represented in Y while the sentences have a high quality. We use BLEU, METEOR, and TER to assess the quality of the sentence (detailed in Section 6.3). Table 6.1 illustrates the input and output of the problem.

This section is organized as follows. First, we describe the overall framework (Section 6.2.1). Next, we describe the pre-processing module of the framework (Section 6.2.2) and the entity-order aware embedding model used in the proposed model (Section 6.2.3). Then, we describe three triple encoder models, including the *adapted BLSTM* model (Section 6.2.4), the *adapted triple encoder* model (Section 6.2.5), and the *proposed GTR-LSTM* model (Section 6.2.6). The same decoder is used for all encoder models, which is described in Section 6.2.7.

6.2.1 Solution Framework

Our solution framework uses an encoder-decoder architecture [21], as illustrated in Figure 6.2. The framework consists of three components, including a *triples pre-processor*, a *target text pre-processor*, and an *encoder-decoder module*.

The triples pre-processor consists of an entity type mapper and a masking module. The entity type mapper maps the subjects and objects in the triples to their types, such that the target sentences are learned based on entity types rather than entities. For example, the input entities in Table 6.1, `John Doe`, `London`, `England`, and `1967-01-10` can be mapped to `PERSON`, `CITY`, `COUNTRY`, and `DATE`, respectively. The mapping has been shown in our experiments to be effective in improving the output quality. The masking module converts each entity into an *entity identifier* (*eid*). The target text pre-processor consists of a text normalizer and a masking module. The text normalizer converts ab-

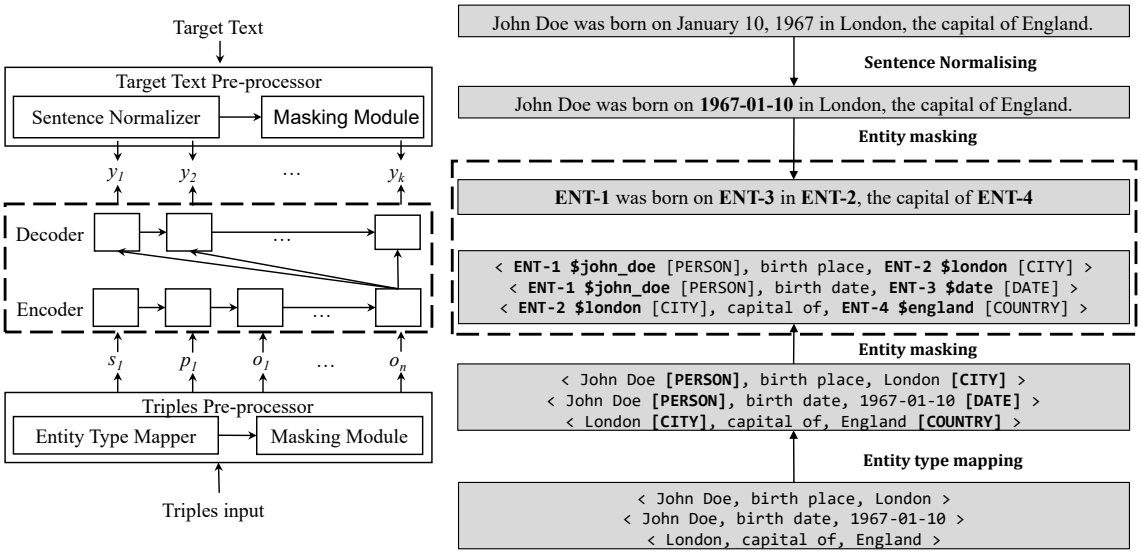


Figure 6.2: Triple-to-text generation based on an encoder-decoder architecture

breviations and dates into the same format as the corresponding entities in the triples. Similar to the masking module of the triples pre-processor, the masking module of the text processor replaces entities in the target sentences by their *eid*s. These pre-processing modules are detailed in Section 6.2.2.

To accommodate the knowledge graph triples as input, in the encoder side, we consider three triple encoder models: (1) an adapted BLSTM encoder; (2) an adapted triple encoder; and (3) the proposed GTR-LSTM triple encoder. The adapted BLSTM encoder linearizes the words in a set of triples as an input sequence, while the adapted triple encoder encodes each triple into a vector representation. The latter model better captures intra-triple relationships but suffers in capturing inter-triple relationships. Considering the native representation of the triples as a graph, our GTR-LSTM encoder captures both intra-triple and inter-triple entity relationships by handling cycles in the input graph and capturing multiple relationships between entities.

To handle multiple relationships between entities, we propose a GTR-LSTM unit that aggregates both an entity and its relationships. Meanwhile, to determine the processing order of the input triples, the proposed GTR-LSTM encoder uses a supervised topological traversal. The traversal takes supervision signals on the processing order from an

entity-order aware embeddings. To learn such entity-order aware embeddings, we train a translation-based graph embedding model over a word-entity graph. The word-entity graph is a KG containing triples from which sentences are to be generated combined with entity-word co-occurrence triples and entity-order triples extracted from a text corpus. Here, an entity-word co-occurrence triple is formed by word-pairs that co-occur within a window in a sentence. An entity-order triple represents a previous relationship between two entities. The entity-order aware embedding models and the proposed GTR-LSTM encoder are detailed in Section 6.2.3 and Section 6.2.6, respectively.

6.2.2 Entity Masking

Entity masking makes our proposed framework generalizes better to unseen entities. This technique addresses the problem of limited training data faced by many natural language generation tasks.

Entity masking replaces entity mentions with *eid*s, *gid*s, and entity types in both the input triples and the target sentences. Here, *eid* is a local entity identifier to differentiate entities in a given input, while *gid* is a global identifier to differentiate entities in the entire dataset. In the case of unseen entities in testing, the *gid* is set to a unique identifier `$unk`. Hence, the entity types play critical roles to differentiate entities in the input. As illustrates in Figure 6.2, the entity `"John Doe"` is replaced by `"ENT-1 $john_doe [PERSON]"`[1] in the input triple and `"ENT-1"` in the target sentence. Here, `"ENT-1"`, `"$john_doe"`, and `"[PERSON]"` are the local identifier, global identifier, and entity type, respectively. To get the entity type, the entity type mapper uses DBpedia lookup API[2]. The API returns several types. The type assigned for each entity is determined by its level in the WordNet [39] hierarchy. We take the type with the highest level in the hierarchy.

In the encoder side, the subject s_n and the object o_n of a triple $t_n = \langle s_n, p_n, o_n \rangle$ are transformed into $\langle eid, gid, type \rangle$, where *eid* is the local identifier, *gid* is the global identifier, and *type* is the entity type extracted by the entity type mapper. Meanwhile, the

[1] To make the following examples more intuitive, we use the first word of the entity mention to represent the entity instead of using *eid* and *gid*. For example, we use `"John"` instead of `"ENT-1 $john_doe"` to represent the entity `"John Doe"` for the following examples.

[2] http://wiki.dbpedia.org/projects/dbpedia-lookup

predicate p_n is preserved, since it indicates the relationship between the subject and the object.

In the decoder side, the entities in the target text are replaced by their corresponding *eid*s. Before masking the entity in the target sentence, we normalize the target sentence to convert the abbreviation and date into the same format as the corresponding entities in the triples. To convert the abbreviation, we use a dictionary of acronyms extracted from Wikipedia[3] [125], while to convert the date we, use regular expressions. Figure 6.2 illustrates the sentence normalization procedure. In this example, the string `"January 10, 1967"` is replaced by `"1967-01-10"` to match the date format on the triple.

Entity matching is not the focus of our study. We use a combination of three string matching methods to find entity mentions in the target sentence: exact matching, *n-gram* matching, and parse tree matching. The exact matching is used to find the exact mention; the *n-gram* matching is used to handle partial matching with the same number of words (e.g., `"John Doe"` and `"John D."`); and parse tree matching is used to find partial matching with a different number of words (e.g., `"John Doe"` and `"John F. Doe"`).

6.2.3 Entity-order Aware Embedding Model

Since we aim to generate sentences based on a given set of triples, it is critical to learn embeddings that preserve the relationships between the words that form the sentences and the entities that form the triples. To train such an embedding model, we adapt a translation-based graph embedding model. A translation-based graph embedding model (e.g., TransE and variants [11, 60, 79, 162]) considers that the embeddings of the object o of a triple should be close to the embeddings of the subject s plus the embeddings of the predicate p, i.e., $\boldsymbol{s} + \boldsymbol{p} \approx \boldsymbol{o}$. This model preserves the structural information of entities, i.e., entities that share similar neighbors in a KG should have a close representation in the embedding space.

To capture relationships between words and entities, we train a translation-based graph embedding model over a word-entity graph constructed as follows (cf. Figure 6.3). The core of the word-entity graph is a knowledge graph containing triples from which

[3]https://github.com/davidsbatista/lexicons

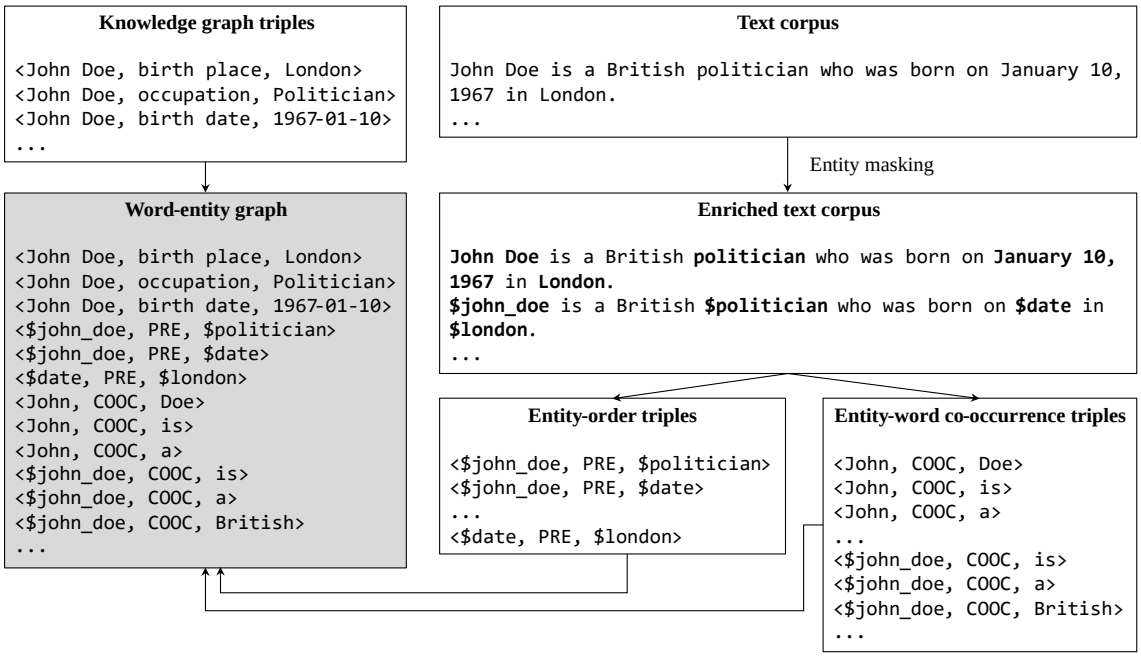


Figure 6.3: Construction of a word-entity graph

sentences are to be generated. This can be a general domain knowledge base, such as DBpedia [3]. We add two types of triples to the core graph. The first is entity-word co-occurrence triples, and the second is entity-order triples.

Both entity-word co-occurrences and entity-order triples are extracted from a text corpus such as Wikipedia. First, we apply entity masking (cf. Section 6.2.2) to the text corpus to enrich them with masked sentences, where entity mentions are replaced by their global identifiers. The masked sentences not only capture relationships between words and entities but also help in entity disambiguation [173]. To collect entity-word co-occurrence triples, we extract word pairs that co-occur in the enriched text corpus within a window W_1 ($W_1 = 5$ in our experiments) and a frequency of co-occurrence of 5. For each extracted pair $\langle w_1, w_2 \rangle$, we create a triple by adding a predefined relationship `COOC`. For example, $\langle$`John,COOC,Doe`$\rangle$ is extracted from the raw sentence and $\langle$`$john_doe,COOC,is`$\rangle$ is extracted from the masked sentence as illustrated in Figure 6.3. To collect entity-order triples, first, we extract a list of *gid*s from left to the right in the masked sentences, e.g., [`$john_doe, $politician, $date, $london`]. Then, we extract the entity pair using a sliding window $W_2 = 2$ from the list. For each ex-

tracted pair $\langle e_1, e_2 \rangle$, we create a triple by adding a predefined relationship `PRE`. For example, $\langle$`$john_doe, PRE, $politician`$\rangle$ is extracted, as illustrated in Figure 6.3. This triple represents a previous relationship between two entities, i.e., the entity `$john_doe` is mentioned before the entity `$politician` in a sentence. We combine these triples with the core graph into a word-entity graph.

Based on the word-entity graph, our embedding model learns the entity and word embeddings by minimizing a margin-based objective function:

$$J_E = \sum_{t_r \in T_r} \sum_{t'_r \in T'_r} \max\left(0, \left[\gamma + f(t_r) - f(t'_r)\right]\right) \tag{6.1}$$

$$T_r = \{\langle s, p, o \rangle | \langle s, p, o \rangle \in G_{WE}\} \tag{6.2}$$

$$T_r{}' = \left\{\langle s', p, o \rangle \mid s' \in Z\right\} \cup \left\{\langle s, p, o' \rangle \mid o' \in Z\right\} \tag{6.3}$$

$$f(t_r) = \left\|\mathbf{M_p}\mathbf{s} + \mathbf{p} - \mathbf{M_p}\mathbf{o}\right\|_2 \tag{6.4}$$

Here, $\|\mathbf{x}\|_2$ is the L2-Norm of vector $\mathbf{x}$, γ is a margin hyperparameter, T_r is the set of valid triples from a word-entity graph G_{WE}, T'_r is the set of corrupted triples, and Z is the set of entities (subject s and object o of a triple) in G_{WE}. The corrupted triples are used as negative samples, which are created by replacing the subject or object of a valid triple in T_r with a random entity. $\mathbf{M_p}$ is a learned matrix to project the subject and object of a triple into a relation space for computing a plausibility score $f(t_r)$.

Since we extract triples from text corpus based on word co-occurrences, the word-entity graph may consist of reflexive (e.g., $\{\langle s, p, o \rangle, \langle o, p, s \rangle\}$), one-to-many (e.g., $\{\langle s, p, o_1 \rangle, \langle s, p, o_2 \rangle\}$), many-to-one (e.g., $\{\langle s_1, p, o \rangle, \langle s_2, p, o \rangle\}$), and many to many relationships (e.g., $\{\langle s_1, p, o_1 \rangle, \langle s_1, p, o_2 \rangle, \langle s_2, p, o_1 \rangle, \langle s_2, p, o_2 \rangle\}$). To handle these relationships, we employ TransR [79] (cf. Eq. (6.4)), where the subject and object embeddings ($\mathbf{s}$ and $\mathbf{o}$) are projected into a vector space by a learned matrix $\mathbf{M_p}$ for each corresponding predicate p.

The advantages of our embedding model are twofold. First, the learned embeddings capture the relationships between words and entities. For the example in Figure 6.3, the embedding model captures the relationship between the entity `$john_doe` and the word `British`. Second, the embeddings preserve the entity order information in a sentence

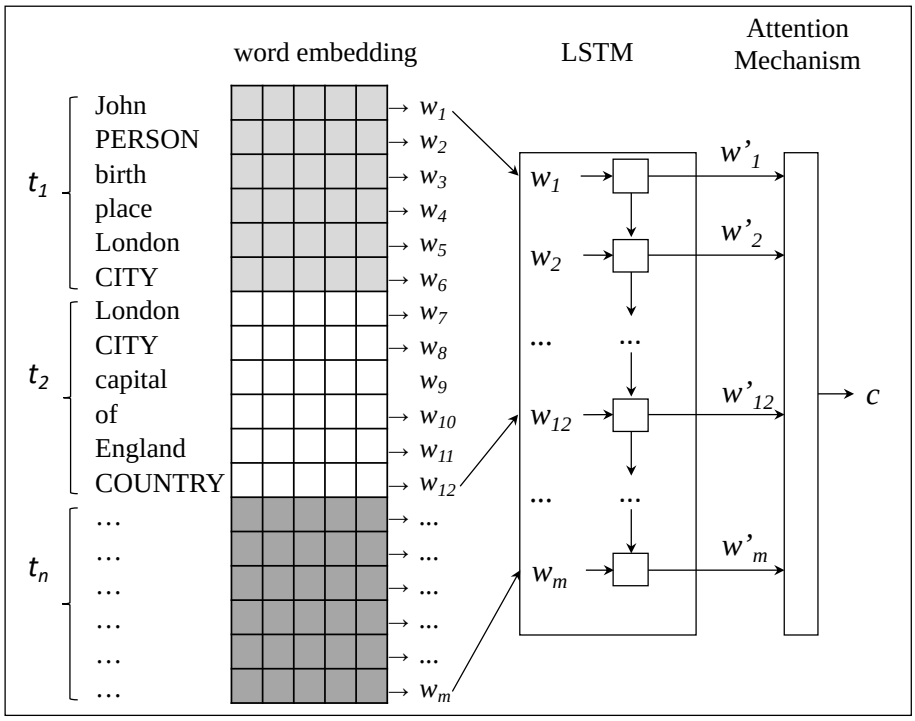


Figure 6.4: Adapted BLSTM encoder

that helps break ties in the topological traversal of the proposed encoder.

The learned embeddings are used to initialized all tested models, including the adapted BLSTM encoder (cf. Section 6.2.4), the adapted triple encoder (cf. Section 6.2.5), and the proposed model (cf. Section 6.2.6).

6.2.4 Adapted BLSTM Encoder

The encoder-decoder model with a bi-directional LSTM (BLSTM) encoder is a sequence-to-sequence learning model [21]. To adapt such a model for our problem, we transform a set of triples input T into a sequence of words (i.e., $T = \{w_1, w_2, ..., w_m\}$), where m is the number of words in a set of triples. As illustrated in Figure 6.4, w_1 is the word embedding of `John`, w_2 is the word embedding of `PERSON`, etc. This sequence forms an input for the encoder. The rest of the model is the same as the encoder-decoder with attention mechanism model [4]. We call this model the *adapted BLSTM encoder*.

6.2.5 Adapted Triple Encoder

The adapted BLSTM encoder suffers in capturing both intra-triple relationships and inter-triple relationships since linearizing a set of triples may discard the graph structure. Next, we further adapt the BLSTM encoder to aggregate the elements of the same triple to retain the intra-triple relationship. We call this the *adapted triple encoder*.

The adaptation is done by grouping the elements of each triple, so the input is represented as $T = \{\langle w_{1,1}, ..., w_{1,j}\rangle, ..., \langle w_{n,1}, ...w_{n,j}\rangle\}$, where $w_{n,j}$ is the embedding of a word in the n-th triple. We use LSTM to compute a hidden state of each triple. We also use an attention mechanism [4] to compute a context vector for the decoder as follows.

$$w'_{t_n} = f_{lstm}(\langle w_{n,1}, w_{n,2}, ..., w_{n,j}\rangle) \tag{6.5}$$

$$\alpha_l = \frac{\exp({h^d}_{k-1}{}^{\intercal} W w'_{t_l})}{\sum_{n=1}^{|T|} \exp({h^d}_{k-1}{}^{\intercal} W w'_{t_n})} \tag{6.6}$$

$$c_k = \sum_{l=1}^{|T|} \alpha_l w'_l \tag{6.7}$$

where w'_{t_n} is a triple vector representation encoded by an LSTM network f_{lstm}, α_l is the attention score of each triple, $|T|$ is the number of triples, ${h^d}_{k-1}$ is the previous hidden state of the decoder, W is a learned parameter, and c_k is the context vector representation for the decoder at time-step k. Figure 6.5(b) illustrates the adapted triple encoder.

6.2.6 GTR-LSTM Triple Encoder

The adapted triple encoder preserves intra-triple relationships. However, it has not considered the structural relationships between entities in different triples. To overcome this limitation, we propose a graph-based triple encoder. We call it the *GTR-LSTM triple encoder*. This encoder takes the input triples in the form of a graph (cf. Figure. 6.6), which preserves the natural structure of the triples. Hence, the proposed encoder captures both intra-triple relationships and inter-triple relationships.

GTR-LSTM differs from existing Graph LSTM [76] and Tree LSTM [147] models in the

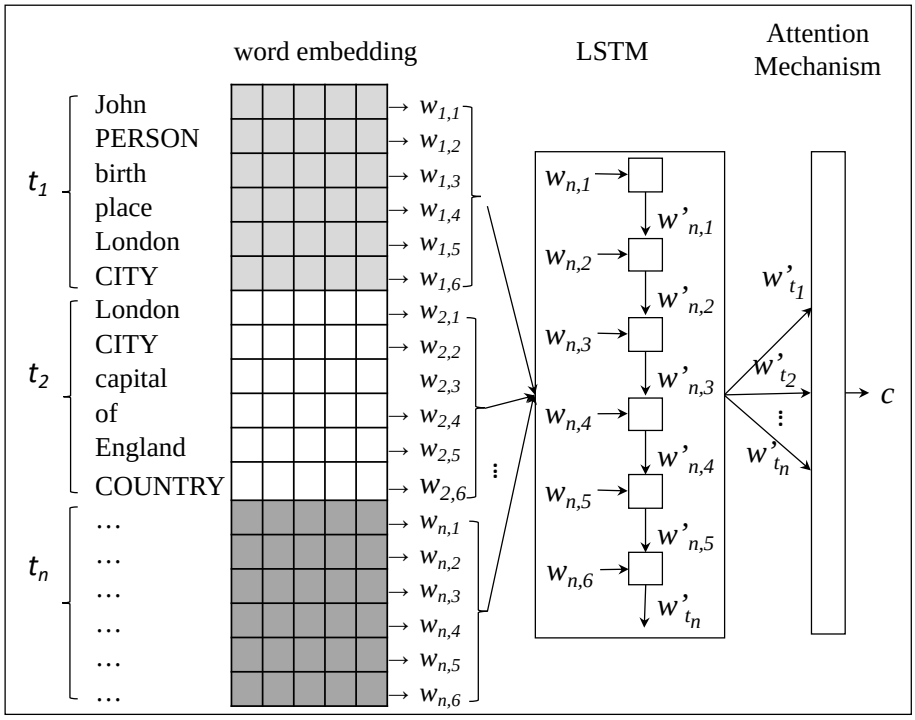


Figure 6.5: Adapted Triple encoders

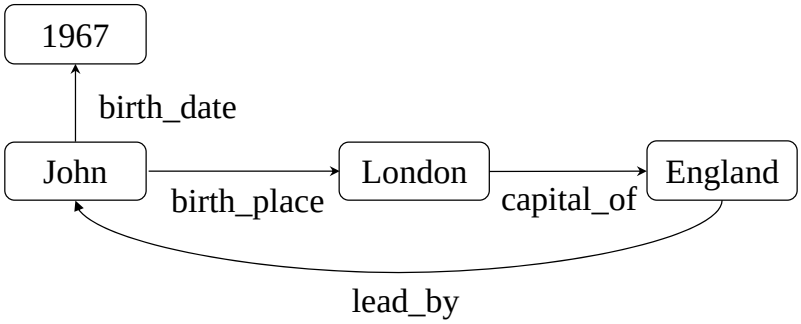


Figure 6.6: A small knowledge graph formed by a set of triples

following aspects. Graph LSTM is proposed for image data. It constructs the graph based on the spatial relationships among super-pixels of an image. Tree LSTM uses the dependency tree as the structure of a sentence. Both models have a predefined relationship between the vertices (Graph LSTM uses spatial relationships: top, bottom, left, or right between super-pixels; Tree LSTM uses dependencies between words in a sentence as the relationship). In contrast, a KG has an open set of relationships between the vertices (i.e., the predicate defines the relationship between entities/vertices), which makes our prob-

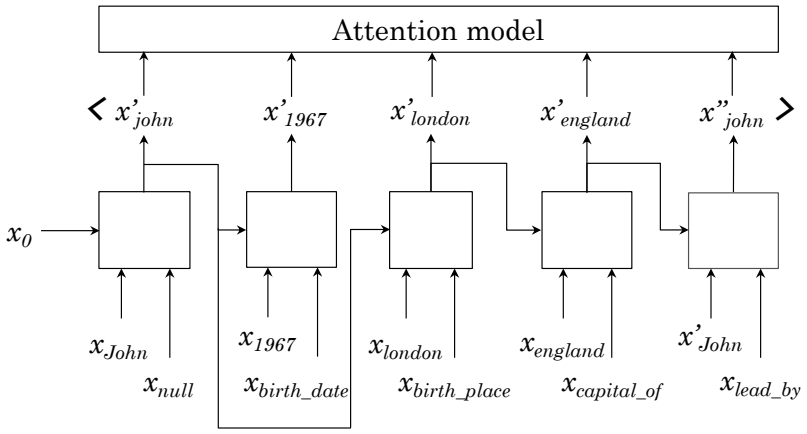


Figure 6.7: GTR-LSTM triple encoder

lem more difficult to model. GTR-LSTM also differs in handling multi-hop relationships between entities in a KG from GCN [68], which is a state-of-the-art graph neural network model. GCN requires multiple layers to capture multi-hop relationships, which makes GCN vulnerable to out-of-memory issues. In contrast, GTR-LSTM requires less memory via recurrent units with a supervised topological traversal algorithm to effectively capture multi-hop relationships.

Our GTR-LSTM triple encoder overcomes the difficulty as follows. It receives a directed graph $G = \langle V, E \rangle$ as the input, where V is a set of vertices that represent entities or literals, and E is a set of directed edges that represent predicates. Here, a vertex (and an edge) may contain multiple words. A vertex (i.e., entity) consists of a local entity identifier *eid*, a global identifier *gid*, and an entity type, while the edge (i.e., predicate) may consist of multiple words, e.g., `"birth date"`. Hence, to represent a vertex (or an edge), we use a linear transformation as follows.

$$x = \tanh(W[w_1; w_2; ...; w_j] + b) \tag{6.8}$$

where $[;]$ indicates vector concatenation, b indicates bias vector, w_j indicates the embeddings of a word in a vertex (or en edge), and W is a learned parameter.

GTR-LSTM Unit

Different from the Graph LSTM, our GTR-LSTM model computes a hidden state by taking into account the processed entity and its edge (the edge pointing to the current entity from the previous entity) to handle multiple relationships between entities in a knowledge graph. Thus, our GTR-LSTM unit (cf. Figure 6.7) receives two inputs, i.e., the entity and its relationship. We propose the following model to compute the hidden state of each GTR-LSTM unit.

$$i_t = \sigma\left(\sum_e \left(U^{ie} x_{te} + W^{ie} x'_{t-1}\right)\right) \tag{6.9}$$

$$f_{te} = \sigma\left(U^f x_{te} + W^f x_{t-1}\right) \tag{6.10}$$

$$o_t = \sigma\left(\sum_e \left(U^{oe} x_{te} + W^{oe} x'_{t-1}\right)\right) \tag{6.11}$$

$$g_t = \tanh\left(\sum_e \left(U^{ge} x_{te} + W^{ge} x'_{t-1}\right)\right) \tag{6.12}$$

$$h_t = \left(h_{t-1} * \sum_e f_{te}\right) + (g_t * i_t) \tag{6.13}$$

$$x'_t = \tanh(h_t) * o_t \tag{6.14}$$

Here, U and W are learned parameter matrices, σ denotes the sigmoid function, $*$ denotes element-wise multiplication, x is the input at the current time-step, and x'_{t-1} is the hidden state of the previously processed vertex. The input gate i determines the weight of the current input. The forget gate f determines the weight of the previous state. The output gate o determines the weight of the cell state forwarded to the next time-step. The state g is the candidate hidden state used to compute the internal memory unit h based on the current input and the previous state. The subscript t is the time-step. The subscript/superscript e is the input element (an entity or a predicate). Following Tree LSTM [147] and Graph LSTM [76], we also use a separate forget gate for each input that allows the GTR-LSTM unit to incorporate information from each input selectively.

Input Graph Visiting Order

To establish an order of feeding the vertices into a GTR-LSTM unit for computing their hidden states, we use a *supervised topological traversal* over the input graph. The traversal decides which hidden state to be computed by sequentially taking a vertex with zero in-degree until all vertices are processed. Our intuition is that the order of entity mentions in a sentence follows the direction of the edge. For example, given a triple ⟨`John, birth place, London`⟩, the graph representation contains a directed edge `birth place` from `John` to `London`, and a common sentence to describe the graph is `"John was born in London"`.

Since the input graph may contain cycles (e.g., Figure 6.6), there could be no vertices with zero in-degree to be visited by the topological traversal. In this case, we need to provide supervision signals to help break ties (i.e., tie-breaking procedure) in the traversal. The supervision signal comes from the learned embeddings, which preserve the information about the order of entity mention in a sentence (cf. Section 6.2.3). Given two vertices v_1 and v_2, the entity to be processed earlier is decided based on their precedence computed by their embeddings. Intuitively, if there is a `PRE` relationship between the entities e_1 and e_2 that correspond to v_1 and v_2, then v_1 should be processed first. Otherwise, we process v_2 first. We use a function f_{pre} to denote the computation of the vertex to be processed first between v_1 and v_2:

$$f_{pre}(v_1, v_2) = \begin{cases} v_1, & \text{if } \cos(\mathbf{M_p w_1} - \mathbf{p_{pre}}, \mathbf{M_p w_2}) \geq 0 \\ v_2, & \text{otherwise} \end{cases} \tag{6.15}$$

where $\mathbf{M_p}$ is the larned projection matrix, $\mathbf{p_{pre}}$ is the embeddings of predicate `PRE` (i.e., a predicate that indicates the relative position between two entities, cf. Section 6.2.3) and $\cos(\mathbf{x}, \mathbf{y})$ is a cosine similarity between two vector $\mathbf{x}$ and $\mathbf{y}$. If there are more than two vertices that have the same smallest in-degree, the function in Eq. (6.15) is run recursively, e.g., $f_{pre}(f_{pre}(v_1, v_2), v_3)$. When a vertex v_i is visited, the hidden states of all adjacent vertices of v_i are computed (or updated if the hidden state of the vertex is already computed in the previous step).

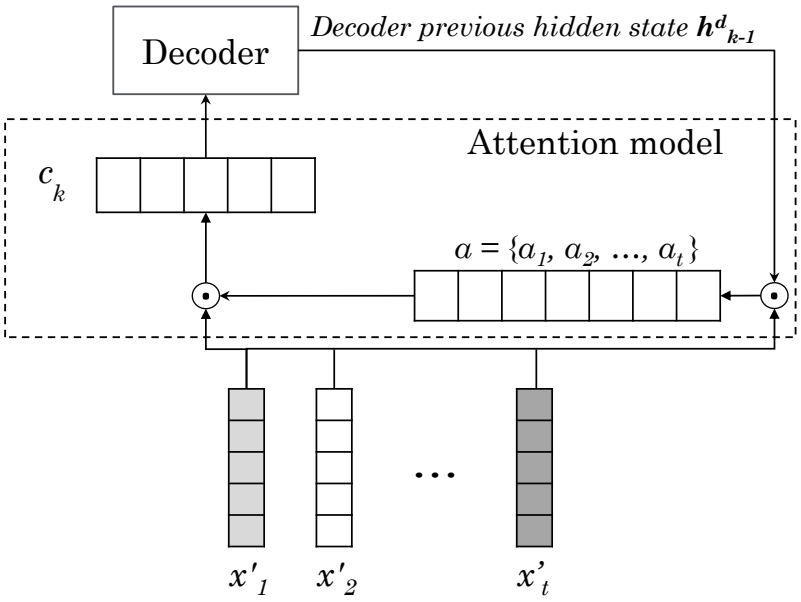


Figure 6.8: GTR-LSTM attention mechanism

Take the graph in Figure. 6.6 as an example. The order of hidden state computation is as follows. The process starts with a vertex with zero in-degree. Because there is no such vertex, we use Eq. (6.15) to determine the next vertex to be processed. Suppose that `John` is chosen as the starting vertex. Then x'_{john} is computed using x_0 as the previous hidden state, and all directed edges started from `John` are removed. Next, x'_{1967} and x'_{London} are computed consecutively (the vertex `1967` is selected by Eq. (6.15), and the vertex `London` is selected in the next topological traversal step) by passing x'_{john} as the previous hidden state. Next, all directed edges started from `london` are removed (no directed edges started from `1967`). In the last step, x''_{john} is updated. Figure. 6.7 illustrates the overall process.

Capturing Global Information of the Input Graph

From Figure. 6.7, we can see that the traversal creates two branches, one ended in x'_{1967}, and the other ended in x''_{john}. After the encoder computes the hidden states of each vertex, x''_{john} does not include the information of x'_{1967} and vice versa. Moreover, the graph can contain cycles that cause difficulty in determining the starting and ending vertices. Our traversal procedure ensures that the hidden states of all vertices are updated based

on their adjacent vertices (local neighbors). To further capture the global information of the graph, we apply an attention model [87] on the GTR-LSTM triple encoder. The attention model takes the hidden states of all vertices computed by the encoder and the previous hidden state of the decoder to compute the context vector of the decoder in each time-step. Figure 6.8 illustrates the attention model of GTR-LSTM. We use the following equation to compute the weights of each vertex.

$$\alpha_t = \frac{\exp({h^d}_{k-1}^{\intercal} W x'_t)}{\sum_{j=1}^{|X|} \exp({h^d}_{k-1}^{\intercal} W x'_j)} \tag{6.16}$$

Here, ${h^d}_{k-1}$ is the previous hidden state of the decoder, $|X|$ is the total number of entities (vertices) in the input triples, W is a learned parameter matrix, x' is the hidden state of a vertex, and $\alpha = \{\alpha_1, \alpha_2, ..., \alpha_t\}$ are the weights of each vertex. Then the context vector of the decoder for each time-step can be computed as follows.

$$c_k = \sum_{t=1}^{|X|} \alpha_t x'_t \tag{6.17}$$

Note that the adapted BLSTM encoder and the adapted triple encoder also use attention mechanisms. However, there are differences between these attention mechanisms. In the BLSTM encoder and the adapted triple encoder, the attention is applied at the word-level and triple-level, respectively. Meanwhile, in the GTR-LSTM encoder, the attention is applied at the entity-level. Attention at the entity-level is more intuitive in selecting the order of entity mention in the target sentence.

6.2.7 Decoder

The decoder of the proposed framework is a standard LSTM. It is trained to generate the output sequence by predicting the next output word y_k conditioned on the hidden state ${h^d}_k$. The current hidden state ${h^d}_k$ is conditioned on the hidden state of the previous time-step ${h^d}_{k-1}$, the output of the previous time-step y_{k-1}, and a context vector c_k. The hidden state and the output of the decoder at time-step k are computed as:

$${h^d}_k = f({h^d}_{k-1}, y_{k-1}, c_k) \tag{6.18}$$

$$\hat{y}_k = softmax(M{h^d}_k) \tag{6.19}$$

Here, $\hat{y}_k$ is the output probability distribution over the vocabulary, f is a single LSTM unit, and M is the hidden-to-output weight matrix. The encoder and the decoder are trained to maximize the conditional log-likelihood:

$$p(S_n \mid T_n) = \sum_{k}^{j=|V|} \sum_{j=1} {y'}_{l,j} \times \log \hat{y}_{k,j} \tag{6.20}$$

Hence, the training objective is to minimize the negative conditional log-likelihood:

$$J = \frac{1}{D} \sum_{d=1}^{D} -\log p(S_n \mid T_n) \tag{6.21}$$

where (S_n, T_n) is a pair of output word sequence and input triple set given for the training, y' is the matrix of the target sentence over the vocabulary V, and $|D|$ is the number of training samples.

6.3 Experiments

We evaluate our framework on two datasets. The first is the dataset from the WebNLG challenge [45]. We call it the *WebNLG* dataset. This dataset contains 25,298 triples-text pairs, with 9,674 unique sets of triples. The dataset consists of a Train+Dev dataset and a Test Unseen dataset. We split Train+Dev into a training set (80%), a development set (10%), and a Seen testing set (10%) (denoted by **Seen** test dataset). The Train+Dev dataset contains triples in ten categories (topics, e.g., astronaut, monument, food, etc.), while the Test Unseen (denoted by **Unseen** test dataset) dataset has five other unseen categories. The maximum number of triples in each triple set is seven. The triples in the Seen test dataset are manually ordered such that the order of the entity in a set of triples follows the order of entity mention in the target sentence. In real applications where the triples are automatically gathered (e.g., for question answering), the entities and triples may not follow their order of appearance in the target sentence. To evaluate the robust-

ness of our model against such scenarios, we randomly shuffled the triple of the Seen dataset (denoted by **Shuffled** test dataset). For the second dataset, we collected data from Wikipedia pages regarding landmarks. We call it the *GKB* dataset. We first extract triples from Wikipedia infoboxes and sentences from the Wikipedia text that contain entities mentioned in the triples. Human annotators then filter out false matches to obtain 1,000 triples-text pairs. This dataset is split into the training and development set (80%) and the testing set (20%) (denoted by **GKB** test dataset). The triples in the GKB dataset are unordered. Table 6.1 illustrates an example of the data pairs of WebNLG and GKB dataset.

We implement the existing models, the adapted model, and the proposed model using Keras. We use three common evaluation metrics, including BLEU [108], METEOR [31], and TER [133]. For the metric computation and significance testing, we use MultEval [24].

6.3.1 Baseline Models

We compare our proposed GTR-LSTM triple encoder with five existing models, including:

- **BLSTM** encoder, which is the adapted Bi-directional LSTM as described in Section 6.2.4.
- **TFF** encoder, which is a triple encoder using feed-forward neural networks [159].
- **SMT** encoder, which is an adapted statistical machine translation [54] model for triple-to-text generation.
- **TLSTM** encoder, which is the adapted triple encoder using LSTM as described in Section 6.2.5.
- **GCN** encoder, which is a state-of-the-art triple encoder based on the graph convolution network [68, 89].

6.3.2 Hyperparameters

We use grid search to find the best hyperparameters for the neural networks. For the embedding model, we choose the embeddings dimensionality among $\{50, 75, 100, 200\}$, the learning rate of the optimizer among $\{0.001, 0.01, 0.1\}$, and the margin γ among $\{1, 5, 10\}$. We train the embedding model with a batch size of 100 and a maximum of 400 epochs. For the proposed encoder model, we use 512 hidden units for both encoder and decoder. We use a 0.5 dropout rate for regularization on both encoder and decoder to avoid overfitting. We train our model on NVIDIA Tesla K40c. We find that using adaptive learning rates for optimization is efficient and leads to faster converge. Thus, we use Adam [67] with a learning rate of 0.0002 instead of stochastic gradient descent. The update of parameters in training is computed using a mini-batch of 64 instances.

6.3.3 Effect of Entity Masking

Table 6.2 shows the overall comparison of model performance. It shows that entity masking gives a consistent performance improvement for all models. Generalizing the input triples and target sentences helps the models to learn the relationships between entities from their types. This is particularly helpful when there is limited training data. We use a combination of exact matching, *n-gram* matching, and parse tree matching to find the entity mentions in the sentence. The entity masking accuracy for WebNLG dataset is 87.15%, while for the GKB dataset is 82.45%.

Entity masking improves the BLEU score of the proposed GTR-LSTM model by 8.5% (from 53.9 on the Entity Unmasking model to 58.5 on the Entity Masking model), 8.7%, 17.6%, and 17.4% on the **Seen**, **Shuffled**, **Unseen**, and **GKB** test datasets, respectively. Using the entity masking not only improves the performance by recognizing the unknown vocabulary via *eid* masking but also improves the running time by requiring a smaller vocabulary.

Table 6.2: Performance comparisons of sentence generation models for generating sentences from an arbitrary-shaped graph

Model	BLEU ↑				METEOR ↑				TER ↓			
	Seen	Shuffled	Unseen	GKB	Seen	Shuffled	Unseen	GKB	Seen	Shuffled	Unseen	GKB
Entity Unmasking												
BLSTM	43.4	43.0	24.0	27.4	35.3	35.1	28.9	26.6	56.0	56.4	69.2	67.0
SMT	41.4	40.2	24.0	27.0	32.5	31.9	28.0	26.7	57.4	58.0	70.1	63.1
TFF	45.5	45.2	26.9	26.3	33.8	33.7	28.8	27.7	52.5	52.5	63.1	60.9
TLSTM	45.0	45.0	27.1	27.3	34.6	34.6	29.4	28.3	51.0	51.2	61.8	59.4
GCN	51.5	51.4	28.9	30.9	36.7	36.1	29.9	29.5	46.6	47.1	62.4	58.9
GTR-LSTM	53.9	53.7	31.2	38.4	37.3	37.1	28.3	30.7	44.6	44.8	59.1	54.5
Entity Masking												
BLSTM	49.5	48.2	27.2	29.1	39.2	38.1	28.6	29.6	49.6	50.0	65.3	66.4
SMT	47.0	46.8	25.7	27.2	36.6	36.1	28.3	30.0	51.6	51.8	62.7	67.5
TFF	46.9	47.1	28.1	30.0	35.5	35.6	31.3	29.5	49.2	49.2	61.2	57.8
TLSTM	51.7	51.6	31.4	32.5	38.6	38.4	31.2	28.9	46.7	46.4	58.7	56.4
GCN	56.1	55.8	32.3	41.3	39.0	38.6	30.9	32.7	42.5	42.8	58.5	52.0
GTR-LSTM	**58.5**	**58.4**	**36.7**	**45.1**	**40.9**	**40.6**	**32.3**	**35.3**	**40.7**	**40.9**	**57.1**	**50.3**

6.3.4 Effect of Models

Table 6.2 also shows that the proposed GTR-LSTM triple encoder achieves a consistent improvement over the baseline models, and the improvement is statistically significant, with $p < 0.05$ based on the t-test of all metrics. We use MultEval to compute the p-value based on approximate randomization [24]. The improvement in the BLEU score indicates that the model reduces the errors in the generated sentence. Our manual inspection confirms this result. The better (lower) TER score suggests that the model generates a more compact output (i.e., better aggregation).

Table 6.3 shows a sample output of all models. From this table, we can see that all baseline models produce sentences that contain wrong relationships between entities (e.g., the BLSTM output contains a wrong relationship "`the elizabeth tower is located in the city of england`"). Moreover, the baseline models generate sentences with a weak aggregation (e.g., `Elizabeth Tower` and `Wembley Stadium` are in separate sentences for TLSTM). The proposed GTR-LSTM model successfully avoids these problems.

Model Training Time

GTR-LSM is slower in training than the baseline models, which is expected as it needs to encode more information. However, its training time is no more than twice as that of any baseline models tested, and the training can complete within one day, which seems reasonable. Meanwhile, the number of parameters trained for GTR-LSTM is up to 59% smaller than those of the baseline models, which saves the space cost for model storage.

6.3.5 Human Evaluation

To complement the automatic evaluation, we conduct human evaluations for all of the masked models. We manage to get ten human annotators. Each of them has studied English for at least ten years and completed education in a fully English environment for at least two years. We provide a website that shows them the triples and the generated text. The annotators are given training on the scoring criteria. We also provide scoring examples. We randomly selected 200 sets of triples along with the output of each model. We only select the sets of triples that contain more than two triples. We use three evaluation metrics [46], including *correctness*, *grammaticality*, and *fluency*. For each pair of a triple set and generated sentences, the annotators are asked to give a score between one to three for each metric.

Correctness is used to measure the semantics of the output. A score of 3 is given to the generated sentences that contain no errors in the relationships between entities; a score of 2 is given to the generated sentences that contain one error in the relationship; a score of 1 is given to the generated sentences that contain more than one errors in the relationships. *Grammaticality* is used to rate the grammatical and spelling errors of the output. Similar to the *correctness* metric, a score of 3 is given to generated sentences with no grammatical and spelling errors; a score of 2 is given to generated sentences with one error; and a score of 1 for the others. The last metric, *fluency*, is used to measure the fluency of the output. The annotators give a score based on the aggregation of the sentences and the existence of sentence repetition. The total time spent for these evaluations is around 300 hours. Table 6.4 shows the results of human evaluations. The results confirm the automatic

Table 6.3: Sample output of the sentence generation models. The error is highlighted in bold.

Input	⟨Elizabeth Tower, location, London⟩, ⟨Wembley Stadium, location, London⟩, ⟨London, capital of, England⟩, ⟨Theresa May, prime minister, England⟩
Reference	london , england is home to wembley stadium and the elizabeth tower. the name of the leader in england is theresa may.
BLSTM	england is lead by theresa may and **is located in the city of london** . the elizabeth tower is located **in the city of england** and is **located in the wembley stadium**.
SMT	wembley stadium is located in **london , elizabeth tower** . theresa may is the leader of **england , england**.
TFF	the elizabeth tower is located in london , england , where **wembley stadium is the leader** and theresa may is the leader.
TLSTM	the wembley stadium is located in london , england . the country is the location of elizabeth tower . **theresa may is the leader of london**.
GCN	**the elizabeth tower, wembley stadium, and london is in england** , where theresa may is the leader .
GTR-LSTM	the wembley stadium and elizabeth tower are both located in london , england . theresa may is the leader of england.

Table 6.4: Human evaluation of sentence generation model for generating sentences from an arbitrary-shaped graph

Model	Seen			Unseen			GKB		
	Correctness	Grammar	Fluency	Correctness	Grammar	Fluency	Correctness	Grammar	Fluency
BLSTM	2.13	2.27	2.14	1.44	1.82	1.58	1.48	1.97	1.74
SMT	1.98	2.04	1.86	1.34	1.52	1.35	1.67	2.09	1.79
TFF	1.82	1.74	1.63	1.31	1.58	1.55	1.72	1.87	2.06
TLSTM	2.32	2.42	2.45	1.52	1.62	1.79	2.13	2.31	2.21
GCN	2.54	2.50	2.31	1.65	1.87	1.91	2.20	2.36	2.15
GTR-LSTM	**2.69**	**2.58**	**2.52**	**1.87**	**2.01**	**2.03**	**2.28**	**2.52**	**2.37**

evaluation in which our proposed model achieves the best scores.

Error Analysis

We further perform a manual inspection of 300 randomly selected output sentences of GTR-LSTM and BLSTM on the Seen and Unseen test data. We find that 35% of BLSTM output contains wrong relationships between entities. In comparison, only 9% of GTR-LSTM output contains such errors. Besides, we find duplicate sub-sentences in the output of GTR-LSTM (14%). The following output is an example: `"beef kway teow is a dish from singapore, where english language is spoken and the leader is tony tan. the leader of singapore is tony tan"`. While the duplication is not wrong, it affects the reading experience. We conjecture that the LSTM in the decoder caused such an issue. We aim to solve this problem in future work.

6.3.6 Ablation Tests

We further perform ablation tests on the proposed encoder model. All compared models use a topological sort to establish an order to encode the input graph. They differ in the tie-breaking procedure as follows.

- GTR-LSTM-**Random** that randomly selects any entity on its tie-breaking procedure.
- GTR-LSTM-**SentenceOrder** that breaks ties based on the order of entity mentions in the target sentence from the input triples. When such an order is unknown, it falls back to a random ordering.
- GTR-LSTM-**TransE** that uses the same tie-breaking procedure as the proposed full GTR-LSM model, but it learns the entity embeddings using TransE [11] instead of TransR in GTR-LSTM-Full.
- GTR-LSTM-**Full** that uses all proposed features, which is detailed in Section 6.2.6

Our proposed embedding model is designed to capture the relationships between both entities and words while preserving the order of entity mention in a sentence. The latter feature helps the tie-breaking procedure of our proposed GTR-LSTM encoder model (detailed in Section 6.2.6). The results of the ablation tests show the effectiveness of the

Table 6.5: Ablation test results of the proposed model

GTRLSTM Model	BLEU ↑				METEOR ↑				TER ↓			
	Seen	Shuffled	Unseen	GKB	Seen	Shuffled	Unseen	GKB	Seen	Shuffled	Unseen	GKB
Entity Unmasking												
TransE	52.8	52.3	30.1	37.0	36.8	35.1	27.9	30.3	45.1	45.5	60.3	56.2
Random	50.9	50.7	24.0	30.8	34.8	34.5	24.2	27.3	46.8	47.4	62.0	58.6
SentenceOrder	54.0	51.6	29.2	37.1	37.3	35.2	27.8	30.6	45.3	46.8	59.8	55.1
Full	53.9	53.7	31.2	38.4	37.3	37.1	28.3	30.7	44.6	44.8	59.1	54.5
Entity Masking												
TransE	56.7	56.4	35.4	44.3	40.2	40.1	31.7	34.7	41.4	41.2	57.9	50.8
Random	55.6	55.4	31.1	40.4	38.8	38.4	30.7	31.8	42.6	42.9	58.3	54.8
SentenceOrder	**58.6**	56.2	35.6	44.7	40.9	39.1	32.0	35.1	**40.6**	42.1	57.8	50.9
Full	58.5	**58.4**	**36.7**	**45.1**	**40.9**	**40.6**	**32.3**	**35.3**	40.7	**40.9**	**57.1**	**50.3**

proposed tie-breaking procedure. Table 6.5 shows that our proposed model outperforms the GTR-LSTM-Random and GTR-LSTM-TransE models consistently. This demonstrates the effectiveness of our model in learning the order of the entities to be mentioned in the target sentences, as neither GTR-LSTM-Random nor GTR-LSTM-TransE considers such order. The GTR-LSTM-TransE model, in particular, only handles one-to-one relationships. Compared to TransE, the performance of TransR in predicting the entity order in a sentence is significantly improved. The precision of TransR is 87.3%, while the precision of TransE is 75.8%.

Comparing with GTR-LSTM-SentenceOrder, GTR-LSTM-Proposed also yields better scores in all three metrics for most cases except for the Seen test data. These results are expected since the triples in the Seen test dataset are ordered according to the entity mentions in the target sentences. However, on the Shuffled and GKB test dataset, the performance of the GTR-LSTM-SentenceOrder model drops substantially. In contrast, our proposed model is robust to such data and generates sentences of higher quality.

6.3.7 Discussions

We also perform further experiments to compare our model with attention-based sequence-to-sequence models, i.e., Transformer model [153]. In machine translation, this model

outperforms LSTM based models. To adapt Transformer for triple-to-text generation, we use the same two strategies as in the LSTM adaptation. The first is transforming the input as a sequence of words, as describes in Section 6.2.4. We call this adaption the *adapted standard transformer*. The second is grouping the elements of each triple, as describes in Section 6.2.5. We call this adaptation the *adapted triple transformer*. However, these models have the same problem as the adaptation of LSTM based models (cf. Section 6.2.1). The experimental results on the Seen test dataset with entity masking confirm that these models fail to capture inter-triple relationships. The adapted standard transformer and the adapted triple transformer achieve 50.7 and 53.2 BLEU scores, respectively.

6.4 Summary

We proposed a novel graph-based triple encoder GTR-LSTM to generate sentence (entity description) from knowledge base triples for enriching a knowledge base. The proposed model maintains the structure of input triples as a graph to optimize the amount of information preserved in the input of the model. The proposed model can handle cycles to capture the global information of a knowledge graph and also handle multiple relationships between entities of a knowledge graph. We improve the encoding process by devising a supervised topological traversal that replaces a random process in the existing encoder, which makes the encoding process more robust. Our supervised traversal procedure helps to encode a knowledge graph in more natural order by taking supervision signals on the processing order from entity-order aware embeddings that trained over a word-entity graph.

The experimental results show that the GTR-LSTM model offers better performance than all the baselines. On the Seen WebNLG dataset, our proposed model outperforms the best existing model, the GCN model, by up to 4.3%, 5.0%, and 4.2% in terms of BLEU, METEOR, and TER scores, respectively. On the GKB dataset, our model outperforms the GCN model by up to 9.2%, 8.0%, and 3.3% in these three metrics, respectively.

Chapter 7
Conclusions

7.1 Summary

IN this thesis, we studied the problem of knowledge base enrichment. Specifically, we study three common methods to enrich a knowledge base, including knowledge bases alignment, relation extraction, and description generation.

In Chapter 3, we study the problem of knowledge bases alignment and propose an embedding model for aligning entities from two different KBs. The proposed model is an embedding-based alignment model built on top of a knowledge graph embedding model that learns entity embeddings to capture the semantic similarity between entities in the different knowledge bases. Embedding-based entity alignment models require both predicate and entity embeddings of two knowledge graphs to fall in the same vector space. For predicate embeddings, our model exploits a *predicate proximity graph*. For entity embedidngs, our model exploits large numbers of attribute triples existing in the knowledge graphs and generates *attribute character embeddings*. The attribute character embedding shifts the entity embeddings from two knowledge graphs into the same space by computing the similarity between entities based on their attributes. We further use a transitivity rule to enrich the number of attributes of an entity to enhance the attribute character embedding. Experiments using real-world knowledge bases show that our proposed model achieves consistent improvements over the baseline models by over 40% in terms of *hits*@1 on the entity alignment task.

In Chapter 4, we study the problem of relation extraction and propose an end-to-end relation extraction for knowledge base enrichment model that integrates the extraction

and canonicalization tasks. Our model thus reduces the error propagation between relation extraction and Named Entity Disambiguation that existing approaches are prone to. Our proposed model is an end-to-end relation extraction model based on a neural encoder-decoder model. We collect high-quality training data by distant supervision with co-reference resolution and paraphrase detection. We propose an n-gram based attention model that captures multi-word entity names in a sentence. Our model employs jointly learned word and entity embeddings to support named entity disambiguation. Finally, our model uses a modified beam search and a triple classifier to help generate high-quality triples. Our model outperforms state-of-the-art baselines by 15.51% and 8.38% in terms of F1 score on two real-world datasets.

In Chapter 5, we study the problem of description generation from a set of triples in the form of a star-shaped-graph and propose an order-agnostic data-to-text generation model. The generated description can be used to enrich information about entities in a knowledge base, which later can be used in many downstream applications. For example, in question answering, the generated sentence can be used to describe the entity in the answer. Previous studies use encoder-decoder frameworks where the encoder treats the input as a linear sequence and uses LSTM to encode the sequence. However, linearizing a set of attributes may not yield the proper order of the attributes, and hence leads the encoder to produce an improper context to generate a description. To handle disordered input, recent studies propose two-stage neural models that use pointer networks to generate a content-plan (i.e., content-planner) and use the content-plan as input for an encoder-decoder model (i.e., text generator). However, in two-stage models, the content-planner may yield an incomplete content-plan, due to missing one or more salient attributes in the generated content-plan. This will, in turn, cause the text generator to generate an incomplete description. To address these problems, we propose a novel attention model that exploits content-plan to highlight salient attributes in a proper order. The challenge of integrating a content-plan in the attention model of an encoder-decoder framework is to align the content-plan and the generated description. We handle this problem by devising a coverage mechanism to track the extent to which the content-plan is exposed in the previous decoding time-step. Hence, it helps our proposed attention

model select the attributes to be mentioned in the description in a proper order. Experimental results show that our model outperforms state-of-the-art baselines by up to 3% and 5% in terms of BLEU score on two real-world datasets, respectively.

In Chapter 6, we study the problem of description generation from arbitrary-shaped graphs as opposed to a star-shaped-graph and propose a novel graph-based triple encoder. Our proposed model computes the hidden state of each entity in a graph that preserves the relationships both within a triple and between the triples, which helps to achieve more accurate sentences. The graph encoder uses entity traversal scheme guided by the entity order in natural sentences that are learned by our entity-order aware translation-based graph embedding model. Experimental results show that the proposed encoder achieves improvements over the baselines by up to 4.3%, 5.0%, and 4.2% in three common metrics BLEU, METEOR, and TER, respectively.

7.2 Future Work

In the problems of knowledge bases alignment, relation extraction, and description generation, the proposed models outperform the state-of-the-art methods. However, the performance of the proposed models can be further improved. Possible future research directions for each of the problems are as follows.

7.2.1 Future Work for Knowledge Bases Alignment

- Our proposed model is built on top of translation based KG embedding models, e.g., TransE. The main challenge addressed is to exploit attribute triples in such an embedding model for entity alignment. Recently, graph neural networks (GNN) KG embedding models are proposed and achieve a better performance than the translation based models in node and graph classification. We plan to integrate the attribute character embedding into such models to achieve better alignment results.

- Auxiliary information, such as predicate and entity descriptions, can be further integrated into the model to compute the predicate add entity similarities.

- Another interesting future work is to adopt an iterative training procedure. For example, the resulting alignments may be used as additional data for an iterative training procedure to get better performance.

7.2.2 Future Work for Relation Extraction

- We plan to generalize the model for extracting new entities and relationships rather than only enriching relationships between existing entities in a knowledge base. This capability is essential since there are lots of new information in the form of new entities, and their relationships can be added to the knowledge base to keep the knowledge base updated.

- We also plan to complement the n-gram attention model with context information such as the surrounding entities in a sentence. In this way, our model can better recognize multi-word entity names disambiguation.

- Another future work is to combine the modified beam search and the triple classifier in a single supervised model to reduce error propagation between them.

7.2.3 Future Work for Description Generation

- We plan to combine the content-plan based attention model into the graph encoder. The content-plan based attention can further help the graph-based encoder in selecting the entities to be mentioned in the target sentence.

- We also plan to employ context similarities for improving the current content-plan extraction that only considers the exact string matching. The combination of context and string similarities can help to extract a more accurate content-plan.

- Another interesting work is to employ a copy mechanism that copies the entity name into the target sentence to replace the masking model. The masking model requires extensive pre-processing work that relies on many external tools such as named entity recognizer, dependency parser, etc., which may propagate some er-

rors into the text generator. Hence, integrating a copy mechanism into the text generator helps reduce error propagation.

This page intentionally left blank.

Bibliography

[1] G. Angeli, P. Liang, and D. Klein, "A simple domain-independent probabilistic approach to generation," in *Proceedings of Empirical Methods in Natural Language Processing*, 2010, pp. 502–512.

[2] G. Angeli, M. J. J. Premkumar, and C. D. Manning, "Leveraging linguistic structure for open domain information extraction," in *Proceedings of Association for Computational Linguistics*, 2015, pp. 344–354.

[3] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. G. Ives, "Dbpedia: A nucleus for a web of open data," in *Proceedings of International Semantic Web Conference*, 2007, pp. 722–735.

[4] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *Proceedings of International Conference on Learning Representations*, 2015.

[5] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni, "Open information extraction from the web," in *Proceedings of International Joint Conference on Artifical intelligence*, 2007, pp. 2670–2676.

[6] J. Bao, D. Tang, N. Duan, Z. Yan, Y. Lv, M. Zhou, and T. Zhao, "Table-to-text: Describing table region with natural language," in *Proceedings of AAAI Conference on Artificial Intelligence*, 2018, pp. 5020–5027.

[7] R. Barzilay and M. Lapata, "Collective content selection for concept-to-text generation," in *Proceedings of Empirical Methods in Natural Language Processing*, 2005, pp. 331–338.

[8] A. Belz, "Automatic generation of weather forecast texts using comprehensive probabilistic generation-space models," *Natural Language Engineering*, vol. 14, no. 4, pp. 431–455, 2008.

[9] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *Journal of machine learning research*, vol. 3, pp. 1137–1155, 2003.

[10] K. Bontcheva and Y. Wilks, "Automatic report generation from ontologies: The miakt approach," in *Proceeding of International Conference on Applications of Natural Language to Information Systems*, 2004, pp. 324–335.

[11] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko, "Translating embeddings for modeling multi-relational data," in *Proceedings of International Conference on Neural Information Processing Systems*, 2013, pp. 2787–2795.

[12] A. Bordes, J. Weston, R. Collobert, and Y. Bengio, "Learning structured embeddings of knowledge bases," in *Proceedings of AAAI Conference on Artificial Intelligence*, 2011, pp. 301–306.

[13] S. Brin, "Extracting patterns and relations from the world wide web," in *Proceedings of World Wide Web and Databases International Workshop*, 1998, pp. 172–183.

[14] Y. Cao, Z. Liu, C. Li, J. Li, and T.-S. Chua, "Multi-channel graph neural network for entity alignment," in *Proceedings of Association for Computational Linguistics*, 2019, pp. 1452–1461.

[15] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. H. Jr., and T. M. Mitchell, "Toward an architecture for never-ending language learning," in *Proceedings of AAAI Conference on Artificial Intelligence*, 2010, pp. 1306–1313.

[16] M. Chaves, M. Silva, and B. Martins, "A geographic knowledge base for semantic web applications," in *Proceedings of Brazilian Symposium on Databases-SBBD*, 2005.

[17] F. Che, D. Zhang, J. Tao, M. Niu, and B. Zhao, "Parame: Regarding neural network parameters as relation embeddings for knowledge graph completion," in *Proceedings of AAAI Conference on Artificial Intelligence*, 2020.

[18] M. Chen, Y. Tian, K.-W. Chang, S. Skiena, and C. Zaniolo, "Co-training embeddings of knowledge graphs and entity descriptions for cross-lingual entity alignment," in *Proceedings of International Joint Conference on Artificial Intelligence*, 2018, pp. 3998–4004.

[19] M. Chen, Y. Tian, M. Yang, and C. Zaniolo, "Multilingual knowledge graph embeddings for cross-lingual knowledge alignment," in *Proceedings of International Joint Conference on Artificial Intelligence*, 2017, pp. 1511–1517.

[20] M. Chen, T. Zhou, P. Zhou, and C. Zaniolo, "Multi-graph affinity embeddings for multilingual knowledge graphs," in *Proceedings of NIPS Workshop on Automated Knowledge Base Construction*, 2017.

[21] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder–decoder for statistical machine translation," in *Proceedings of Empirical Methods in Natural Language Processing*, 2014, pp. 1724–1734.

[22] J. Christensen, S. Soderland, O. Etzioni *et al.*, "Semantic role labeling for open information extraction," in *Proceedings of Workshop on Formalisms and Methodology for Learning by Reading*, 2010, pp. 52–60.

[23] P. Cimiano, J. Lüker, D. Nagel, and C. Unger, "Exploiting ontology lexica for generating natural language texts from rdf data," in *Proceedings of European Workshop on Natural Language Generation*, 2013, pp. 10–19.

[24] J. H. Clark, C. Dyer, A. Lavie, and N. A. Smith, "Better hypothesis testing for statistical machine translation: Controlling for optimizer instability," in *Proceedings of Association for Computational Linguistics*, 2011, pp. 176–181.

[25] K. Clark and C. D. Manning, "Deep reinforcement learning for mention-ranking coreference models," in *Proceedings of Empirical Methods in Natural Language Processing*, 2016, pp. 2256–2262.

[26] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *Journal of Machine Learning Research*, vol. 12, pp. 2493–2537, 2011.

[27] L. D. Corro and R. Gemulla, "Clausie: clause-based open information extraction," in *Proceedings of International Conference on World Wide Web*, 2013, pp. 355–366.

[28] L. Cui, F. Wei, and M. Zhou, "Neural open information extraction," in *Proceedings of Association for Computational Linguistics*, 2018, pp. 407–413.

[29] R. Das, S. Dhuliawala, M. Zaheer, L. Vilnis, I. Durugkar, A. Krishnamurthy, A. Smola, and A. McCallum, "Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning," in *Proceedings of International Conference on Learning Representations*, 2018.

[30] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society*, vol. 39, pp. 1–22, 1977.

[31] M. J. Denkowski and A. Lavie, "Meteor 1.3: Automatic metric for reliable optimization and evaluation of machine translation systems," in *Proceedings of Workshop on Statistical Machine Translation*, 2011, pp. 85–91.

[32] T. Dettmers, P. Minervini, P. Stenetorp, and S. Riedel, "Convolutional 2d knowledge graph embeddings," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[33] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 2019, pp. 4171–4186.

[34] P. A. Duboue and K. R. McKeown, "Empirically estimating order constraints for content planning in generation," in *Proceedings of Association for Computational Linguistics*, 2001, pp. 172–179.

[35] P. A. Duboue and K. R. McKeown, "Content planner construction via evolutionary algorithms and a corpus-based fitness function," in *Proceedings of International Conference on Natural Language Generation*, 2002, pp. 89–96.

[36] P. A. Duboue and K. R. McKeown, "Statistical acquisition of content selection rules for natural language generation," in *Proceedings of Empirical Methods in Natural Language Processing*, 2003, pp. 121–128.

[37] D. Duma and E. Klein, "Generating natural language from linked-data: Unsupervised template extraction," in *Proceedings of International Conference on Computational Semantics*, 2013, pp. 83–94.

[38] A. Fader, S. Soderland, and O. Etzioni, "Identifying relations for open information extraction," in *Proceedings of Empirical Methods in Natural Language Processing*, 2011, pp. 1535–1545.

[39] C. Fellbaum, *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

[40] D. Ferrucci, A. Levas, S. Bagchi, D. Gondek, and E. T. Mueller, "Watson: beyond jeopardy!" *Artificial Intelligence*, vol. 199, pp. 93–105, 2013.

[41] J. L. Fleiss, "Measuring nominal scale agreement among many raters." *Psychological bulletin*, vol. 76, no. 5, p. 378, 1971.

[42] L. Galárraga, G. Heitz, K. Murphy, and F. M. Suchanek, "Canonicalizing open knowledge bases," in *Proceedings of International Conference on Information and Knowledge Management*, 2014, pp. 1679–1688.

[43] L. Galarraga, C. Teflioudi, K. Hose, and F. M. Suchanek, "Fast rule mining in ontological knowledge bases with amie," *Very Large Data Bases*, vol. 24, pp. 707–730, 2015.

[44] J. Ganitkevitch, B. V. Durme, and C. Callison-Burch, "Ppdb: The paraphrase database," in *Proceedings of North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2013, pp. 758–764.

[45] C. Gardent, A. Shimorina, S. Narayan, and L. Perez-Beltrachini, "Creating training corpora for nlg micro-planners," in *Proceedings of Association for Computational Linguistics*, 2017, pp. 179–188.

[46] C. Gardent, A. Shimorina, S. Narayan, and L. Perez-Beltrachini, "The webnlg challenge: Generating text from rdf data," in *Proceedings of International Conference on Natural Language Generation*, 2017, pp. 124–133.

[47] K. Gashteovski, R. Gemulla, and L. D. Corro, "Minie: Minimizing facts in open information extraction," in *Proceedings of Empirical Methods in Natural Language Processing*, 2017, pp. 2620–2630.

[48] A. Grycner and G. Weikum, "Poly: Mining relational paraphrases from multilingual sentences," in *Proceedings of Empirical Methods in Natural Language Processing*, 2016, pp. 2183–2192.

[49] J. Gu, Z. Lu, H. Li, and V. O. Li, "Incorporating copying mechanism in sequence-to-sequence learning," in *Proceedings of Association for Computational Linguistics*, 2016, pp. 1631–1640.

[50] C. Gulcehre, S. Ahn, R. Nallapati, B. Zhou, and Y. Bengio, "Pointing the unknown words," in *Proceedings of Association for Computational Linguistics*, 2016, pp. 140–149.

[51] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[52] L. He, K. Lee, O. Levy, and L. Zettlemoyer, "Jointly predicting predicates and arguments in neural semantic role labeling," in *Proceedings of Association for Computational Linguistics*, 2018, pp. 364–369.

[53] Q. He, L. Wu, Y. Yin, and H. Cai, "Knowledge-graph augmented word representations for named entity recognition," in *Proceedings of AAAI Conference on Artificial Intelligence*, 2020.

[54] H. Hoang and P. Koehn, "Design of the moses decoder for statistical machine translation," in *Proceeding of Software Engineering, Testing, and Quality Assurance for Natural Language Processing*, 2008, pp. 58–65.

[55] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, pp. 1735–1780, 1997.

[56] J. Hoffart *et al.*, "Robust disambiguation of named entities in text," in *Proceedings of Empirical Methods in Natural Language Processing*, 2011, pp. 782–792.

[57] J. Hoffart, F. M. Suchanek, K. Berberich, and G. Weikum, "Yago2: A spatially and temporally enhanced knowledge base from wikipedia," *Artificial Intelligence*, vol. 194, pp. 28–61, 2013.

[58] R. Hoffmann, C. Zhang, and D. S. Weld, "Learning 5000 relational extractors," in *Proceedings of Association for Computational Linguistics*, 2010, pp. 286–295.

[59] G. Huang, Z. Liu, L. v. d. Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of Conference on Computer Vision and Pattern Recognition*, 2017, pp. 2261–2269.

[60] G. Ji, K. Liu, S. He, and J. Zhao, "Knowledge graph completion with adaptive sparse transfer matrix," in *Proceedings of AAAI Conference on Artificial Intelligence*, 2016, pp. 985–991.

[61] G. Ji, K. Liu, S. He, and J. Zhao, "Distant supervision for relation extraction with sentence-level attention and entity descriptions," in *Proceedings of AAAI Conference on Artificial Intelligence*, 2017, pp. 3060–3066.

[62] R. Jia and P. Liang, "Data recombination for neural semantic parsing," in *Proceedings of Association for Computational Linguistics*, 2016, pp. 12–22.

[63] X. Jiang, Q. Wang, P. Li, and B. Wang, "Relation extraction with multi-instance multi-label convolutional neural networks," in *Proceedings the International Conference on Computational Linguistics*, 2016, pp. 1471–1480.

[64] L. Jin, L. Song, Y. Zhang, K. Xu, W. yun Ma, and D. Yu, "Relation extraction exploiting full dependency forests," in *Proceedings of AAAI Conference on Artificial Intelligence*, 2020.

[65] M. Kathuria, C. Nagpal, and N. Duhan, "Journey of web search engines: Milestones, challenges & innovations," *International Journal of Information Technology and Computer Science*, vol. 12, pp. 47–58, 2016.

[66] J. Kim and R. J. Mooney, "Generative alignment and semantic parsing for learning from ambiguous supervision," in *Proceedings International Conference on Computational Linguistics*, 2010, pp. 543–551.

[67] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," in *Proceedings of International Conference on Learning Representations*, 2015.

[68] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proceedings of International Conference on Learning Representations*, 2017.

[69] A. Kobren, T. Logan, S. Sampangi, and A. McCallum, "Domain specific knowledge base construction via crowdsourcing," in *Proceedings of Neural Information Processing Systems Workshop on Automated Knowledge Base Construction*, 2014.

[70] N. Kolitsas, O.-E. Ganea, and T. Hofmann, "End-to-end neural entity linking." in *Proceedings of Conference on Computational Natural Language Learning*, 2018, pp. 519–529.

[71] D. Kurita, B. Roengsamut, K. Kuwabara, and H.-H. Huang, "Knowledge base refinement with gamified crowdsourcing," in *Proceedings of Asian Conference on Intelligent Information and Database Systems*, 2016, pp. 33–42.

[72] R. Lebret, D. Grangier, and M. Auli, "Neural text generation from structured data with application to the biography domain," in *Proceedings of Empirical Methods in Natural Language Processing*, 2016, pp. 1203–1213.

[73] C. Li, P. Zhao, V. S. Sheng, X. Xian, J. Wu, and Z. Cui, "Refining automatically extracted knowledge bases using crowdsourcing," *Computational Intelligence and Neuroscience*, vol. 2017, pp. 1–17, 2017.

[74] R. Li, S. Wang, F. Zhu, and J. Huang, "Adaptive graph convolutional neural networks," in *Proceedings of AAAI Conference on Artificial Intelligence*, 2018, pp. 3546–3553.

[75] Y. Li, G. Long, T. Shen, T. Zhou, L. Yao, H. Huo, and J. Jiang, "Self-attention enhanced selective gate with entity-aware embedding for distantly supervised relation extraction," in *Proceedings of AAAI Conference on Artificial Intelligence*, 2020.

[76] X. Liang, X. Shen, J. Feng, L. Lin, and S. Yan, "Semantic object parsing with graph lstm," in *Proceedings of European Conference on Computer Vision*, 2016, pp. 125–143.

[77] X. V. Lin, R. Socher, and C. Xiong, "Multi-hop knowledge graph reasoning with reward shaping," in *Proceedings of Empirical Methods in Natural Language Processing*, 2018, pp. 3243–3253.

[78] Y. Lin, Z. Liu, and M. Sun, "Neural relation extraction with multi-lingual attention," in *Proceedings of Association for Computational Linguistics*, 2017, pp. 34–43.

[79] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu, "Learning entity and relation embeddings for knowledge graph completion," in *Proceedings of AAAI Conference on Artificial Intelligence*, 2015, pp. 2181–2187.

[80] Y. Lin, S. Shen, Z. Liu, H. Luan, and M. Sun, "Neural relation extraction with selective attention over instances," in *Proceedings of Association for Computational Linguistics*, 2016, pp. 2124–2133.

[81] T. Liu, S. Ma, Q. Xia, F. Luo, B. Chang, and Z. Sui, "Hierarchical encoder with auxiliary supervision for table-to-text generation: Learning better representation for tables," in *Proceedings of AAAI Conference on Artificial Intelligence*, 2019, pp. 6786–6793.

[82] T. Liu, K. Wang, L. Sha, Z. Sui, and B. Chang, "Table-to-text generation by structure-aware seq2seq learning," in *Proceedings of AAAI Conference on Artificial Intelligence*, 2018, pp. 4881–4888.

[83] W. Liu, P. Zhou, Z. Zhao, Z. Wang, Q. Ju, H. Deng, and P. Wang, "K-bert: Enabling language representation with knowledge graph," in *Proceedings of AAAI Conference on Artificial Intelligence*, 2020.

[84] Z. Liu, C. Chen, L. Li, J. Zhou, X. Li, L. Song, and Y. Qi, "Geniepath: Graph neural networks with adaptive receptive paths," *Proceedings of AAAI Conference on Artificial Intelligence*, pp. 4424–4431, 2019.

[85] W. Lu and H. T. Ng, "A probabilistic forest-to-string model for language generation from typed lambda calculus expressions," in *Proceedings of Empirical Methods in Natural Language Processing*, 2011, pp. 1611–1622.

[86] W. Lu, H. T. Ng, and W. S. Lee, "Natural language generation with tree conditional random fields," in *Proceedings of Empirical Methods in Natural Language Processing*, 2009, pp. 400–409.

[87] T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *Proceedings of Empirical Methods in Natural Language Processing*, 2015, pp. 1412–1421.

[88] C. Malaviya, C. Bhagavatula, A. Bosselut, and Y. Choi, "Commonsense knowledge base completion with structural and semantic context," in *Proceedings of AAAI Conference on Artificial Intelligence*, 2020.

[89] D. Marcheggiani and L. Perez-Beltrachini, "Deep graph convolutional encoders for structured data to text generation," in *Proceedings of International Conference on Natural Language Generation*, 2018, pp. 1–9.

[90] Mausam, "Open information extraction systems and downstream applications," in *Proceedings of International Joint Conference on Artificial Intelligence*, 2016, pp. 4074–4077.

[91] Mausam, M. Schmitz, S. Soderland, R. Bart, and O. Etzioni, "Open language learning for information extraction," in *Proceedings of Empirical Methods in Natural Language Processing*, 2012, pp. 523–534.

[92] K. R. McKeown, D. A. Jordan, S. Pan, J. Shaw, and B. A. Allen, "Language generation for multimedia healthcare briefings," in *Proceedings of Conference on Applied Natural Language Processing*, 1997, pp. 277–282.

[93] H. Mei, M. Bansal, and M. R. Walter, "What to talk about and how? selective generation using lstms with coarse-to-fine alignment," in *Proceedings of North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2016, pp. 720–730.

[94] C. Mellish, A. Knott, J. Oberlander, and M. O'Donnell, "Experiments using stochastic search for text planning," in *Proceedings of ACL Workshop on Natural Language Generation*, 1998, pp. 98–107.

[95] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proceedings of International Conference on Neural Information Processing Systems*, 2013, pp. 3111–3119.

[96] D. Milne and I. H. Witten, "An effective, low-cost measure of semantic relatedness obtained from wikipedia links," in *Proceedings of AAAI Workshop on Wikipedia and Artificial Intelligence*, 2008, pp. 25–30.

[97] M. Mintz, S. Bills, R. Snow, and D. Jurafsky, "Distant supervision for relation extraction without labeled data," in *Proceedings of Joint Conference of Association for Computational Linguistics and International Joint Conference on Natural Language Processing of the AFNLP*, 2009, pp. 1003–1011.

[98] M. Miwa and M. Bansal, "End-to-end relation extraction using lstms on sequences and tree structures," in *Proceedings of Association for Computational Linguistics*, 2016.

[99] N. Nakashole, G. Weikum, and F. M. Suchanek, "Patty: A taxonomy of relational patterns with semantic types," in *Proceedings of Empirical Methods in Natural Language Processing*, 2012, pp. 1135–1145.

[100] T. Nayak and H. T. Ng, "Effective modeling of encoder-decoder architecture for joint entity and relation extraction," in *Proceedings of AAAI Conference on Artificial Intelligence*, 2020.

[101] A.-C. N. Ngomo and S. Auer, "Limes: a time-efficient approach for large-scale link discovery on the web of data," in *Proceedings of International Joint Conference on Artificial Intelligence*, 2011, pp. 2312–2317.

[102] D. Q. Nguyen, T. D. Nguyen, D. Q. Nguyen, and D. Q. Phung, "A novel embedding model for knowledge base completion based on convolutional neural network," in *Proceedings of North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2018, pp. 327–333.

[103] T. H. Nguyen and R. Grishman, "Relation extraction: Perspective from convolutional neural networks," in *Proceedings of Workshop on Vector Space Modeling for Natural Language Processing*, 2015, pp. 39–48.

[104] M. Nickel, L. Rosasco, and T. A. Poggio, "Holographic embeddings of knowledge graphs," in *Proceedings of AAAI Conference on Artificial Intelligence*, 2016, pp. 1955–1961.

[105] M. Nickel, V. Tresp, and H.-P. Kriegel, "Factorizing yago: scalable machine learning for linked data," in *Proceedings of International Conference on World Wide Web*, 2012, pp. 271–280.

[106] X. Niu, S. Rong, H. Wang, and Y. Yu, "An effective rule miner for instance matching in a web of data," in *Proceedings of International Conference on Information and Knowledge Management*, 2012, pp. 1085–1094.

[107] H. Pal *et al.*, "Demonyms and compound relational nouns in nominal open ie," in *Proceedings of Workshop on Automated Knowledge Base Construction*, 2016, pp. 35–39.

[108] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of Association for Computational Linguistics*, 2002, pp. 311–318.

[109] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of Empirical Methods in Natural Language Processing*, 2014, pp. 1532–1543.

[110] M. Pershina, M. Yakout, and K. Chakrabarti, "Holistic entity matching across knowledge graphs," in *Proceedings of International Conference on Big Data*, 2015, pp. 1585–1590.

[111] M. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," in *Proceedings of North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2018, pp. 2227–2237.

[112] R. Puduppully, L. Dong, and M. Lapata, "Data-to-text generation with content selection and planning," in *Proceedings of AAAI Conference on Artificial Intelligence*, 2019, pp. 6908–6915.

[113] P. Qin, X. Wang, W. Chen, chunyun zhang, W. Xu, and W. Y. Wang, "Generative adversarial zero-shot relational learning for knowledge graphs," in *Proceedings of AAAI Conference on Artificial Intelligence*, 2020.

[114] Y. Raimond, C. Sutton, and M. B. Sandler, "Automatic interlinking of music datasets on the semantic web." in *Proceedings of Linking Data on the Web Workshop*, 2008.

[115] D. Ravichandran and E. Hovy, "Learning surface text patterns for a question answering system," in *Proceedings of the Annual Meeting of The Association for Computational Linguistics*, 2002, pp. 41–47.

[116] E. Reiter and R. Dale, *Building natural language generation systems*. Cambridge University Press, 2000.

[117] S. Riedel, L. Yao, and A. McCallum, "Modeling relations and their mentions without labeled text," in *Proceedings of European Conference on Machine Learning and Knowledge Discovery in Databases*, 2010, pp. 148–163.

[118] S. Riedel, L. Yao, A. McCallum, and B. M. Marlin, "Relation extraction with matrix factorization and universal schemas," in *Proceedings of North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2013, pp. 74–84.

[119] E. Rivas and S. R. Eddy, "A dynamic programming algorithm for rna structure prediction including pseudoknots," *Journal of Molecular Biology*, vol. 285, no. 5, pp. 2053–2068, 1999.

[120] B. Roengsamut, K. Kuwabara, and H.-H. Huang, "Toward gamification of knowledge base construction," in *Proceedings of International Symposium on Innovations in Intelligent SysTems and Applications*, 2015, pp. 1–7.

[121] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 696–699, 1988.

[122] C. D. Sa, A. Ratner, C. Ré, J. Shin, F. Wang, S. Wu, and C. Zhang, "Incremental knowledge base construction using deepdive," *Very Large Data Bases Journal*, vol. 26, no. 1, pp. 81–105, 2017.

[123] F. Scharffe, F. L. Yanbin, and C. Zhou, "Rdf-ai: an architecture for rdf datasets matching, fusion and interlink," in *Proceedings of IJCAI Workshop on Identity and Reference in Knowledge Representation*, 2009.

[124] M. S. Schlichtkrull, T. N. Kipf, P. Bloem, R. van den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *Proceedings of International Conference on Extended Semantic Web Conference*, 2018, pp. 593–607.

[125] A. S. Schwartz and M. A. Hearst, "A simple algorithm for identifying abbreviation definitions in biomedical text," *Biocomputing*, pp. 451–462, 2002.

[126] A. See, P. J. Liu, and C. D. Manning, "Get to the point: Summarization with pointer-generator networks," in *Proceedings of Association for Computational Linguistics*, 2017, pp. 1073–1083.

[127] I. V. Serban, A. García-Durán, C. Gulcehre, S. Ahn, S. Chandar, A. Courville, and Y. Bengio, "Generating factoid questions with recurrent neural networks: The 30M factoid question-answer corpus," in *Proceedings of Association for Computational Linguistics*, 2016, pp. 588–598.

[128] L. Sha, L. Mou, T. Liu, P. Poupart, S. Li, B. Chang, and Z. Sui, "Order-planning neural text generation from structured data," in *Proceedings of AAAI Conference on Artificial Intelligence*, 2018, pp. 5414–5421.

[129] C. Shang, Y. Tang, J. Huang, J. Bi, X. He, and B. Zhou, "End-to-end structure-aware convolutional networks for knowledge base completion," in *Proceedings of AAAI Conference on Artificial Intelligence*, 2019, pp. 3060–3067.

[130] W. Shen, J. Wang, and J. Han, "Entity linking with a knowledge base: Issues, techniques, and solutions," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, pp. 443–460, 2015.

[131] Y. Shinyama and S. Sekine, "Preemptive information extraction using unrestricted relation discovery," in *Proceedings of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2006, pp. 304–311.

[132] N. C. Shu, B. C. Housel, R. W. Taylor, S. P. Ghosh, and V. Y. Lum, "Express: a data extraction, processing, and restructuring system," *ACM Transactions on Database Systems*, vol. 2, pp. 134–174, 1977.

[133] M. Snover, B. Dorr, R. Schwartz, L. Micciulla, and J. Makhoul, "A study of translation edit rate with targeted human annotation," in *Proceeding of Association for Machine Translation in the Americas*, 2006, pp. 223–231.

[134] R. Socher, D. Chen, C. D. Manning, and A. Y. Ng, "Reasoning with neural tensor networks for knowledge base completion," in *Proceedings of International Conference on Neural Information Processing Systems*, 2013, pp. 926–934.

[135] D. Sorokin and I. Gurevych, "Context-aware representations for knowledge base relation extraction." in *Proceedings of Empirical Methods in Natural Language Processing*, 2017, pp. 1784–1789.

[136] C. Stadler, J. Lehmann, K. Höffner, and S. Auer, "Linkedgeodata: A core for a web of spatial open data," *Semantic Web*, vol. 3, p. 333–354, 2012.

[137] G. Stanovsky, J. Michael, I. Dagan, and L. Zettlemoyer, "Supervised open information extraction," in *Proceedings of North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2018, pp. 885–895.

[138] G. Stoica, O. Stretcu, A. Platanios, T. Mitchell, and B. Poczos, "Contextual parameter generation for knowledge graph link prediction," in *Proceedings of AAAI Conference on Artificial Intelligence*, 2020.

[139] F. M. Suchanek, G. Kasneci, and G. Weikum, "Yago: a core of semantic knowledge," in *Proceedings of International Conference on World Wide Web*, 2007, pp. 697–706.

[140] F. M. Suchanek, M. Sozio, and G. Weikum, "SOFIE: a self-organizing framework for information extraction," in *Proceedings of International Conference on World Wide Web*, 2009, pp. 631–640.

[141] Z. Sun, W. Hu, and C. Li, "Cross-lingual entity alignment via joint attribute-preserving embedding," in *Proceedings of International Semantic Web Conference*, 2017, pp. 628–644.

[142] Z. Sun, W. Hu, Q. Zhang, and Y. Qu, "Bootstrapping entity alignment with knowledge graph embedding," in *Proceedings of International Joint Conference on Artificial Intelligence*, 2018, pp. 4396–4402.

[143] Z. Sun, C. Wang, W. Hu, M. Chen, J. Dai, W. Zhang, and Y. Qu, "Knowledge graph alignment network with gated multi-hop neighborhood aggregation," in *Proceedings of AAAI Conference on Artificial Intelligence*, 2020.

[144] M. Surdeanu and M. Ciaramita, "Robust information extraction with perceptrons," in *Proceedings of the Automatic Content Extraction Workshop*, 2007.

[145] M. Surdeanu, J. Tibshirani, R. Nallapati, and C. D. Manning, "Multi-instance multi-label learning for relation extraction," in *Proceedings of Empirical Methods in Natural Language Processing*, 2012, pp. 455–465.

[146] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proceedings of Neural Information Processing Systems*, 2014, pp. 3104–3112.

[147] K. S. Tai, R. Socher, and C. D. Manning, "Improved semantic representations from tree-structured long short-term memory networks," in *Proceedings of International Joint Conference on Natural Language Processing*, 2015, pp. 1556–1566.

[148] R. Takanobu, T. Zhang, J. Liu, and M. Huang, "A hierarchical framework for relation extraction with reinforcement learning," in *Proceedings of AAAI Conference on Artificial Intelligence*, 2019, pp. 7072–7079.

[149] Z. Tu, Y. Liu, L. Shang, X. Liu, and H. Li, "Neural machine translation with reconstruction," in *Proceedings of AAAI Conference on Artificial Intelligence*, 2017.

[150] Z. Tu, Z. Lu, Y. Liu, X. Liu, and H. Li, "Modeling coverage for neural machine translation," in *Proceedings of Association for Computational Linguistics*, 2016, pp. 76–85.

[151] K. van Deemter, E. Krahmer, and M. Theune, "Real versus template-based natural language generation: A false opposition?" *Computational Linguistics*, vol. 31, no. 1, pp. 15–24, 2005.

[152] S. Vashishth, S. Sanyal, V. Nitin, N. Agrawal, and P. Talukdar, "Interacte: Improving convolution-based knowledge graph embeddings by increasing feature interactions," in *Proceedings of AAAI Conference on Artificial Intelligence*, 2020.

[153] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of Neural Information Processing Systems*, 2017, pp. 5998–6008.

[154] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *Proceddings of International Conference on Learning Representations*, 2018.

[155] A. P. B. Veyseh, F. Dernoncourt, M. Thai, D. Dou, and T. Nguyen, "Multi-view consistency for relation extraction via mutual information and structure prediction," in *Proceedings of AAAI Conference on Artificial Intelligence*, 2020.

[156] O. Vinyals, S. Bengio, and M. Kudlur, "Order matters: Sequence to sequence for sets," in *Proceedings of International Conference on Learning Representations*, 2016.

[157] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Proceedings of International Conference on Neural Information Processing Systems*, 2015, pp. 2692–2700.

[158] J. Volz, C. Bizer, M. Gaedke, and G. Kobilarov, "Discovering and maintaining links on the web of data," in *Proceedings of International Semantic Web Conference*, 2009, pp. 650–665.

[159] P. Vougiouklis, H. Elsahar, L.-A. Kaffee, C. Gravier, F. Laforest, J. Hare, and E. Simperl, "Neural wikipedian: Generating textual summaries from knowledge base triples," *Journal of Web Semantics*, vol. 52-53, pp. 1 – 15, 2018.

[160] D. Vrandecic and M. Krötzsch, "Wikidata: a free collaborative knowledgebase," *Communications of the ACM*, vol. 57, pp. 78–85, 2014.

[161] P. Wang, Q. Wu, C. Shen, A. Dick, and A. van den Hengel, "Fvqa: Fact-based visual question answering," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 2413–2427, 2018.

[162] Z. Wang, J. Zhang, J. Feng, and Z. Chen, "Knowledge graph embedding by translating on hyperplanes," in *Proceedings of AAAI Conference on Artificial Intelligence*, 2014, pp. 1112–1119.

[163] Z. Wang, Q. Lv, X. Lan, and Y. Zhang, "Cross-lingual knowledge graph alignment via graph convolutional networks," in *Proceedings of Conference on Empirical Methods in Natural Language Processing*, 2018, pp. 349–357.

[164] Z. Wang and J. Li, "Text-enhanced representation learning for knowledge graph," in *Proceedings of International Joint Conference on Artificial Intelligence*, 2016, pp. 1293–1299.

[165] P. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[166] S. J. Wiseman, S. M. Shieber, and A. S. M. Rush, "Challenges in data-to-document generation," in *Proceedings of Empirical Methods in Natural Language Processing*, 2017, pp. 2253–2263.

[167] Q. Wu, C. Shen, P. Wang, A. Dick, and A. v. d. Hengel, "Image captioning and visual question answering based on attributes and external knowledge," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1367–1381, 2018.

[168] Y. Wu, X. Liu, Y. Feng, Z. Wang, R. Yan, and D. Zhao, "Relation-aware entity alignment for heterogeneous knowledge graphs," in *Proceedings of International Joint Conference on Artificial Intelligence*, 2019, pp. 5278–5284.

[169] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *arXiv preprint arXiv:1901.00596*, 2019.

[170] Y. Xiao, C. Tan, Z. Fan, Q. Xu, and W. Zhu, "Joint entity and relation extraction with a hybrid transformer and reinforcement learning based model," in *Proceedings of AAAI Conference on Artificial Intelligence*, 2020.

[171] R. Xie, Z. Liu, J. Jia, H. Luan, and M. Sun, "Representation learning of knowledge graphs with entity descriptions," in *Proceedings of AAAI Conference on Artificial Intelligence*, 2016, pp. 2659–2665.

[172] K. Xu, liwei wang, M. Yu, Y. Feng, Y. Song, Z. Wang, and D. Yu, "Cross-lingual knowledge graph alignment via graph matching neural network," in *Proceeddings of Association for Computational Linguistics*, 2019, pp. 3156–3161.

[173] I. Yamada, H. Shindo, H. Takeda, and Y. Takefuji, "Joint learning of the embedding of words and entities for named entity disambiguation." in *Proceedings of Conference on Computational Natural Language Learning*, 2016, pp. 250–259.

[174] Z. Yang, P. Blunsom, C. Dyer, and W. Ling, "Reference-aware language models," in *Proceedings of Conference on Empirical Methods in Natural Language Processing*, 2017, pp. 1850–1859.

[175] R. Ye, X. Li, Y. Fang, H. Zang, and M. Wang, "A vectorized relational graph convolutional network for multi-relational network alignment." in *Proceedings of International Joint Conference on Artificial Intelligence*, 2019, pp. 4135–4141.

[176] D. Zeng, K. Liu, Y. Chen, and J. Zhao, "Distant supervision for relation extraction via piecewise convolutional neural networks," in *Proceedings of Empirical Methods in Natural Language Processing*, 2015, pp. 1753–1762.

[177] D. Zeng, K. Liu, S. Lai, G. Zhou, and J. Zhao, "Relation classification via convolutional deep neural network," in *Proceedings International Conference on Computational Linguistics*, 2014, pp. 2335–2344.

[178] X. Zeng, S. He, D. Zeng, K. Liu, S. Liu, and J. Zhao, "Learning the extraction order of multiple relational facts in a sentence with reinforcement learning," in *Proceedings The Conference on Empirical Methods in Natural Language Processing*, 2019, pp. 367–377.

[179] X. Zeng, D. Zeng, S. He, K. Liu, and J. Zhao, "Extracting relational facts by an end-to-end neural model with copy mechanism," in *Proceedings of Association for Computational Linguistics*, 2018, pp. 506–514.

[180] J. Zhang, X. Shi, J. Xie, H. Ma, I. King, and D. yan Yeung, "Gaan: Gated attention networks for learning on large and spatiotemporal graphs," in *Proceedings of The Conference on Uncertainty in Artificial Intelligence*, 2018, pp. 339–349.

[181] Z. Zhang, J. Cai, Y. Zhang, and J. Wang, "Learning hierarchy-aware knowledge graph embeddings for link prediction," in *Proceedings of AAAI Conference on Artificial Intelligence*, 2020.

[182] Z. Zhang, F. Zhuang, H. Zhu, Z. Shi, H. Xiong, and Q. He, "Relational graph neural network with hierarchical attention for knowledge graph completion," in *Proceedings of AAAI Conference on Artificial Intelligence*, 2020.

[183] Z. Zhang, X. Shu, B. Yu, T. Liu, J. Zhao, Q. Li, and L. Guo, "Distilling knowledge from well-informed soft labels for neural relation extraction," in *Proceedings of AAAI Conference on Artificial Intelligence*, 2020.

[184] G. Zhou, J. Su, J. Zhang, and M. Zhang, "Exploring various knowledge in relation extraction," in *Proceedings of the Annual Meeting of The Association for Computational Linguistics*, 2005, pp. 427–434.

[185] P. Zhou, J. Xu, Z. Qi, H. Bao, Z. Chen, and B. Xu, "Distant supervision for relation extraction with hierarchical selective attention," *Neural Networks*, vol. 108, pp. 240–247, 2018.

[186] H. Zhu, R. Xie, Z. Liu, and M. Sun, "Iterative entity alignment via joint knowledge embeddings," in *Proceedings of International Joint Conference on Artificial Intelligence*, 2017, pp. 4258–4264.

[187] C. Zhuang and Q. Ma, "Dual graph convolutional networks for graph-based semi-supervised classification," in *Proceedings of International World Wide Web Conferences*, 2018, pp. 499–508.