Learning with Graphs in Natural Language Processing

Submitted by

Zhijiang GUO

Thesis Advisor

Prof. Wei LU

Information Systems Technology and Design

A thesis submitted to the Singapore University of Technology and Design in fulfillment of the requirement for the degree of Doctor of Philosophy

2021

PhD Thesis Examination Committee

TEC Chair:	Prof. Ngai-Man Cheung
Main Advisor:	Prof. Wei Lu
Internal TEC member 1:	Prof. Alexander Binder
Internal TEC member 2:	Prof. Jun Liu
External TEC member 1:	Prof. Shay Cohen (University of Edinburgh)

"*There never was a sun in the sky over me. It's always night. But not dark. I had something in place of the sun. Maybe not as bright, but enough for me. Enough so I was able to live in the night like it was day. You can't be afraid of losing something if you never had.*"

Keigo Higashino

Abstract

Information Systems Technology and Design

Doctor of Philosophy

Learning with Graphs in Natural Language Processing

by Zhijiang GUO

Graph is a ubiquitous structure in natural language processing (NLP), which describes a collection of entities, represented as nodes, and their pairwise relationships, represented as edges. Many sentence-level meaning representations employ directed, acyclic graphs as the underlying formalism, while most tree-based syntactic representations can also be regarded as graphs. In this thesis, we mainly focus on predicting graphs from texts, and integrating graphs for downstream tasks, such as natural language generation and relation extraction.

We first investigate how to predict the corresponding semantic graph from the input sentence. Specifically, we focus on the Abstract Meaning Representation (AMR), which is a directed, labelled graph. Such a graph can capture rich structural information, and is useful for semantics-related tasks. AMR parsing is a challenging task as it requires the parser to learn to predict a large number of concepts and relations based on relatively sparse training data. We introduce a simple yet effective transition-based AMR parser, which is able to conduct the search in a refined search space based on a new compact AMR graph and an improved oracle.

Then we explore how to leverage graph structures for natural language generation. The key challenge in graph encoding is how to efficiently learn an effective graph representation. We introduce dense connections and the dynamic fusion mechanism to graph convolutional networks (GCNs). With the help of these mechanisms, the GCN model is able to capture rich local and non-local information associated with a graph. Two novel weight sharing strategies are further developed to reduce memory usage and model complexity.

Apart from language generation, relation extraction models also benefit from incorporating graph structural information. A key challenge in graph-based models is the error propagation. To better address this issue, we propose a soft pruning strategy that is able to distil the information associated with the dependency graph. For document-level and bio-medical texts, the error propagation issue becomes severe due to the low parsing accuracy. We further develop models based on latent structure induction, which treats the dependency graph as a latent variable and induces it in an end-to-end fashion.

Keyword: Graph Representation, Semantic Parsing, Natural Language Generation, Relation Extraction

Publications

* Denotes equal contribution.

- Yan Zhang*, **Zhijiang Guo***, Zhiyang Teng, Wei Lu, Shay B. Cohen, Zuozhu Liu and Lidong Bing. 2020. Lightweight, Dynamic Graph Convolutional Networks for AMR-to-Text Generation. In *EMNLP*.
- **Zhijiang Guo***, Guoshun Nan*, Wei Lu and Shay B. Cohen. 2020. Learning Latent Forests for Medical Relation Extraction. In *IJCAI*.
- Guoshun Nan*, **Zhijiang Guo***, Ivan Sekulic and Wei Lu. Reasoning with Latent Structure Refinement for Document-Level Relation Extraction. 2020 In *ACL*.
- **Zhijiang Guo***, Yan Zhang*, Zhiyang Teng and Wei Lu. Densely Connected Graph Convolutional Networks for Graph-to-Sequence Learning. 2019 In *TACL*.
- **Zhijiang Guo***, Yan Zhang* and Wei Lu. Attention Guided Graph Convolutional Networks for Relation Extraction. 2019. In *ACL*.
- **Zhijiang Guo** and Wei Lu. 2018. Better Transition-Based AMR Parsing with a Refined Search Space. In *EMNLP*.

Acknowledgements

First and foremost, I would like to thank Professor Wei Lu for giving me the opportunity to be a PhD student in the StatNLP group and under his supervision. I used to be a PhD student working on network analysis under the EPD pillar. Back then, I knew nothing about natural language processing or machine learning. Without Professor Lu's help, I won't be able to start the journey of doing research about human language over these four years. I am tremendously grateful for his continuous support, for giving me the freedom to pursue a variety of research directions, and for frequently reminding me to do good research. Professor Lu is the person who is sincerely serious about research. He is always the role model of my academic career.

I am very grateful to have had the opportunity to visit University of Edinburgh under Shay and work on the tree-to-graph transducer project with Giorgio, Frank and Johanna since the summer of 2019. Shay not only has an insightful view about the field, but also understands the details of the models including how to implement them. I really enjoy the time and learn a lot from Shay when he shares his ideas and jokes during the group meeting. I also want to thank Giorgio for the inspirational weekly discussion, even though I have never met him in person. He is also an extremely kind and supportive advisor. He is like an older friend of mine (if he doesn't mind) and I can talk with him about everything, such as science fiction and arts.

Also, I would like to thank Doctor Zhiyang Teng, who is a significant collaborator and a teacher to me. He is literally the most diligent and knowledgeable PhD student I have ever met. Moreover, he is always willing to help me, especially during my most difficult days. Zhiyang has answered tons of my research questions, taught me how to implement various models and more importantly, how to be an independent researcher. I would like to thank my other two significant collaborators Doctor Yan Zhang and Doctor Guoshun Nan for countless discussions and their efforts in implementing our proposed models and running lots of experiments. Moreover, they offered me a place to stay when I was homeless in Singapore.

During my PhD, I am honored to be a member of StatNLP group. I learned a great lot from these great researchers. For StatNLP group, I would like to thank Hao Li, Allan Jie, Raymond Susanto, Yanyan Zou, Yan Zhang, Bill Lin, Thilini Cooray, Boyuan Zhuang, Guoshun Nan and Ivan Sekulic for discussing and brainstorming. Participating in activities in Cohort and Edinburgh NLP group is a wonderful and truly enriching experience. I would like to thank Bailin Wang, Zheng Zhao, Chunchuan Lyu, Zaixiang Zheng, Yanpeng Zhao, Hao Zheng, Biao Zhang, Yumo Xu, Jiangming Liu, Yang Liu, Bowen Li, Xinchi Chen and Ronald Cardenas. In particular, Bailin—who encouraged me to visit Edinburgh and help me a lot after I came to the U.K. I also have fond memories in Bayes Center where I shared an office with Zaixiang Zheng and Yu Yang. I could not have hoped for a more inspiring working environment.

Outside of the NLP community, I have been extremely lucky to be surrounded by many great friends. Zhizhao Chen and the little gang back in Sun-Yat sen University including Chen Zhou, Pingping Zhang, Xiuwen Cai and Yiran Ge (alphabetical order) are my closest friends for many years. I would like to thank them for their love and support. They keep pulling me out from the stressful PhD life and make me feel that I am not lonely when I am alone in other countries for these years. Sihan Wang is my best friend after we joined the PhD program together back in September 2016. He taught me lots of meta-skills to survive in this tough PhD journey. I would also like to thank Professor Jianxi Luo for recruiting me as a PhD student under EPD pillar and Sin Chee for helping me with all kinds of stuff in ISTD. I also like to thank other PhD students in SUTD for helping me out when I started the journey, to name a few, Yuejun He, Bowen Yan, Binyang Song, Yang Jie and WeiKang Wu.

Lastly, and more importantly, I would like to thank my family. My mother, she always supports me to pursue my dream even though she went through a serious car accident. Without her unconditional love and support, I am not able to achieve any goals in my career. Fan Wu, she is not only my partner, my best friend but also the person I admire for her intelligence, concentration and hard work. Thank you for enduring my absence, and for always being or having been there for me.

Contents

List of Figures

List of Tables

List of Abbreviations

NLP	**N**atural **L**anguage **P**rocessing
AMR	**A**bstract **M**eaning **R**epresentation
MR	**M**eaning **R**epresentation
NLG	**N**atural **L**anguage **G**eneration
IE	**I**nformation **E**xtraction
GCNs	**G**raph **C**onvolutional **N**etworks
DCGCNs	**D**ensely **C**onnected **G**raph **C**onvolutional **N**etworks
AGGCNs	**A**ttention **G**uided **G**raph **C**onvolutional **N**etworks
LDGCNs	**L**ightweight, **D**ynamic **G**raph **C**onvolutional **N**etworks
GRNs	**G**raph **R**ecurrent **N**etworks
GGNNs	**G**ated **G**raph **N**eural **N**etworks
SANs	**S**elf-**A**ttention **N**eworks
LSTM	**L**ong **S**ort-**T**erm **M**emory
GRU	**G**ated **R**ecurrent **U**nit
CNN	**C**onvolutional **N**eural **N**etworks
LSR	**L**atent **S**tructure **R**efinement

To my family and friends, for their endless support.

Chapter 1

Introduction

1.1 Graph Structures in NLP

Graph structure is all around us: from social networks to the world wide web, street maps, knowledge bases used in search engines, and even chemical molecules. Graphs also play a significant role in natural language processing (NLP) as they are able to capture richer structural information than sequences. In this thesis, we mainly focus on two types of graphs in NLP, including the semantic graph and dependency graph.

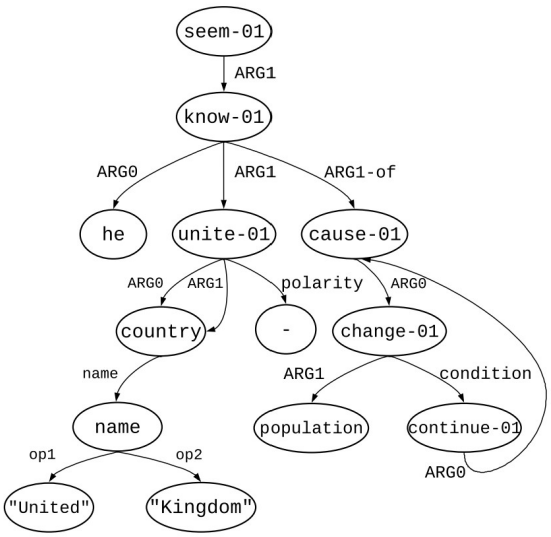


FIGURE 1.1: Example AMR for a sentence: *It seems that he knows under the population changes (if it continues) that U.K. will not unite.*

The semantic graph that this thesis works with is Abstract Meaning Representation (AMR; Banarescu et al. 2013). AMR is a formalism that captures the semantics of a sentence with a rooted directed graph, in which nodes represent the concepts and edges represent the relations between concepts. An example AMR corresponds to the sentence "*It seems that he knows under the population changes (if it continues) that U.K. will not unite.*" is illustrated in Figure 1.1.

AMR encodes information about semantic relations, named entities, co-reference, negation and modality. For example, the subgraph in the box corresponds to the name entity "*U.K.*" in the input sentence. One distinctive aspect of AMR annotation is the lack of explicit alignments between nodes in the graph (concepts) and words in the sentences. As shown in Figure 1.1, the stop word "*that*" does not appear in the AMR while the word "*U.K*" corresponds to multiple

concepts. As AMR abstracts away from details of surface realization, it is potentially beneficial in many semantic related NLP tasks, including question answering (Mitra and Baral, 2016), text summarization (Liao, Lebanoff, and Liu, 2018), and machine translation (Song et al., 2019b).

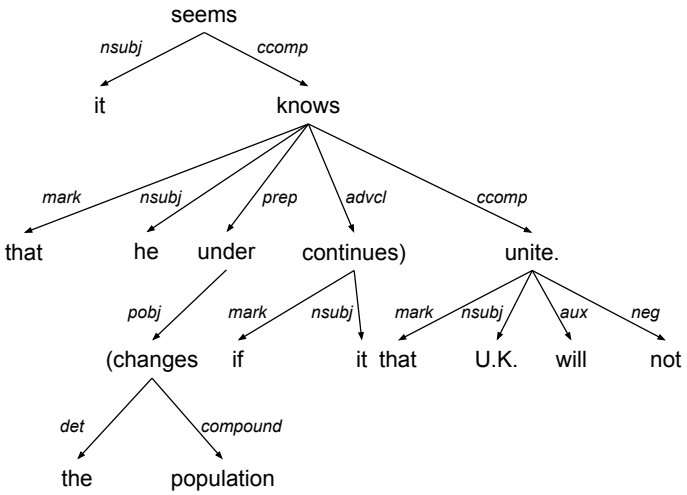


FIGURE 1.2: Example dependency graph for the same sentence: *It seems that he knows under the population changes (if it continues) that U.K. will not unite.* Punctuation is merged for simplicity.

On the other hand, dependency structure of sentences shows which words depend on (modify or are arguments of) which other words. These binary asymmetric relations between the words are called dependencies and are depicted as edges going from the head (or governor) to the dependent (or modifier). Usually these dependencies form a tree, which can also be viewed as directed, single-head graph (Jurafsky and Martin, 2020). Dependency graph is often typed with the name of grammatical relations (e.g. subject, prepositional object, apposition). An example of a dependency tree corresponds to the same sentence for AMR is shown in Figure 1.2. Unlike the AMR, all words in the input sentence are nodes in the dependency graph. As the dependency graph is able to capture long-range syntactic relations that are obscure from the surface form alone, it is beneficial in various downstream tasks, such as semantic role labeling (Marcheggiani and Titov, 2017), name entity recognition (Jie and Lu, 2019) and sentiment analysis (Huang and Carley, 2019)

1.2 Motivation

Motivated by the fact that graph structures are able to capture rich structural information, we focus on predicting graphs from texts and incorporate graphs for natural language generation and relation extraction. Specifically, we investigate how to build an effective parser to obtain the AMR graph, develop an efficient structural encoder to encode graphs (e.g. AMR and dependency graphs) and model dependency graphs for better relational learning. We raise several research questions for us to exploit and try to address them in this thesis.

1.2.1 Predicting Semantic Graph

Though AMR encodes rich semantic-level information that is potentially beneficial to downstream tasks, it is non-trivial to obtain the AMR from the plain text (i.e. AMR parsing). The AMR parser is required to predict not only concepts, which consist of predicates, lemmas, named entities, wiki-links and co-references, but also a large number of relation types based on relatively sparse training data (Peng et al., 2017b). Given the challenges, many state-of-the-art AMR parsers employ various external resources and adopt a pipeline approach (Flanigan et al., 2014; Lyu and Titov, 2018; Zhang et al., 2019a).

The transition-based approach (Yamada and Matsumoto, 2003; Nivre, 2003) has been popular among many NLP tasks, including syntactic parsing (Chen and Manning, 2014), named entity recognition (Lample et al., 2016), and semantic parsing (Cheng et al., 2017). Different from the graph-based approach for structured prediction (e.g., conditional random fields; Lafferty, McCallum, and Pereira 2001), such an approach is able to maintain a good balance between efficiency and accuracy (Nivre and McDonald, 2008), and has achieved state-of-the-art results on a number of tasks (Swayamdipta et al., 2016; Shi, Huang, and Lee, 2017). While the transition-based approach is promising for structured prediction, existing transition-based AMR parsers (Damonte, Cohen, and Satta, 2016; Ballesteros and Al-Onaizan, 2017) still cannot attain the state-of-the-art results on such a task. Thus, we raise the following research question:

- *Can we further improve existing transition-based AMR parsing systems?*

We observe that the key to the development of an effective transition-based system is a properly defined search space, and argue that the search space used in existing transition systems needs to be refined (Guo and Lu, 2018). We propose to conduct the search in a refined search space based on a new compact AMR graph and an improved oracle. Our AMR parser achieves the state-of-the-art performance on various datasets with minimal additional information.

1.2.2 Encoding Graph for Natural Language Generation

After obtaining the semantic graph, the main challenge is how to encode such a structure for downstream tasks, such as natural language generation. Prior efforts (Konstas et al., 2017) use a sequence encoder to encode the input graph. As shown in Figure 1.3, they first use the depth-first traversal to linearize the AMR graph to obtain a sequence of tokens. Then a bi-directional LSTM network (Schuster and Paliwal, 1997) is used to encode the sequence. Sequence encoders may lose structural information from the original graph because they require linearization of the input graph.

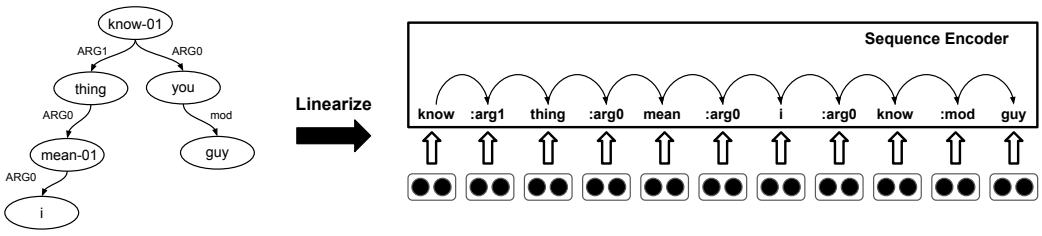


FIGURE 1.3: Sequence Encoder.

To better encode the graph structure, recent efforts (Beck, Haffari, and Cohn, 2018; Song et al., 2018a; Ribeiro, Gardent, and Gurevych, 2019) leverage recurrence based graph neural

networks to encode the structural information of graphs. As shown in Figure 1.4, recurrent graph encoder is able to encode graph-level semantics by performing information exchange between connected nodes based on the gating mechanism (Hochreiter and Schmidhuber, 1997).

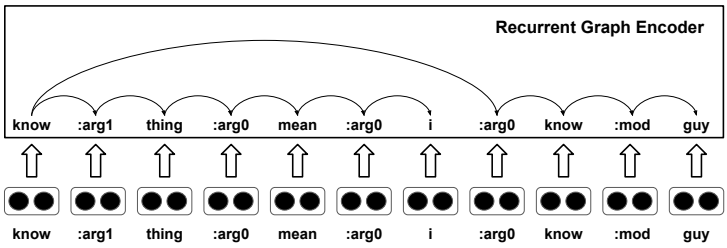


FIGURE 1.4: Recurrent Graph Encoder.

More recently, Self-Attention Networks (SANs;Vaswani et al. 2017) have been explored to capture global dependencies. As shown in Figure 1.5, SANs associate each node with other nodes such that interactions between any two nodes can be modeled in the graph. Still, this approach ignores the structure of the original graph. Zhu et al. (2019) and Cai and Lam (2020b) further propose structured SANs that incorporate additional neural components to encode the structural information of the input graph.

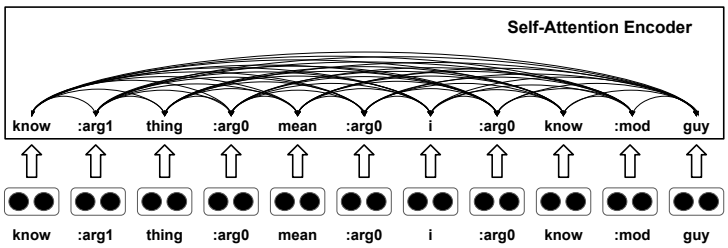


FIGURE 1.5: Self-Attention Encoder.

Compared with recurrent neural networks, convolutional architectures are highly parallelizable and are more amenable to hardware acceleration (Gehring et al., 2017). On the other hand, convolutional operations are more computationally efficient than self attention operations because the computation of attention weights scales quadratically while convolutions scale linearly with respect to the input length (Wu et al., 2019a).

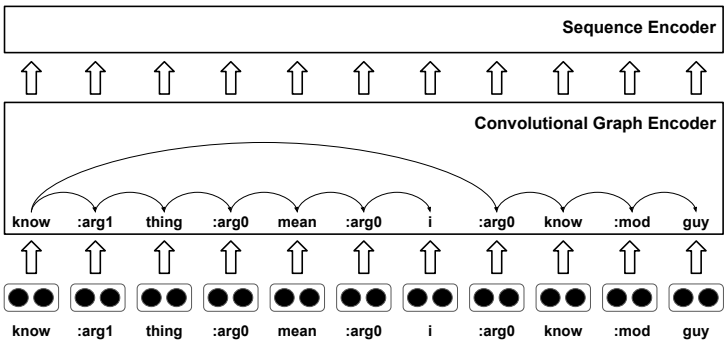


FIGURE 1.6: Mixed Encoder.

It is therefore worthwhile to explore the possibility of applying convolutional networks as the structural encoder. Graph convolutional networks (GCNs;Kipf and Welling 2017) are variants of convolutional neural networks (CNNs;LeCun and Bengio 1995) that operate directly on graphs. In GCNs, the representation of each node is iteratively updated based on those of its adjacent nodes in the graph through a local information propagation scheme. However, existing GCN models are still confined to relatively shallow architectures (two layers), as the performance of the GCN model drops when the number of layers increases (Xu et al., 2018). Also, the locality property of existing graph convolutions preclude efficient non-local information propagation (Cai and Lam, 2020b).

In order to capture non-local interactions for larger graphs, recent works (Bastings et al., 2017; Damonte and Cohen, 2019) leverage a mixed encoder by adding a sequence encoder on top of the convolutional graph encoder, as shown in Figure 1.6. However, both recurrent graph encoder and self attention encoder outperform mix encoder empirically. Adding an additional recurrent neural network on top of the GCN model also offsets the benefit that convolutional architectures are highly parallelizable. Thus, we raise a research question:

- *Can we train a deeper GCN model that is able to better capture non-local information for learning the graph representation?*

To address this research question, we introduce dense connections to GCNs and train a multi-layer GCN model with a large depth for learning a better graph representation (Guo et al., 2019). We further develop the dynamic fusion mechanism, which enables the GCN model to integrate high-order information for capturing non-local information (Zhang et al., 2020).

1.2.3 Modelling Graph for Relation Extraction

Apart from natural language generation, relation extraction models also benefit from incorporating structural information. For example, dependency graphs can be used to capture non-local syntactic relations that are obscure from the surface form alone for better relation extraction (Zhang, Qi, and Manning, 2018). A key challenge in graph-based models is the error propagation that noises are introduced to the model due to parsing errors, offsetting much of the benefit that relation extraction models can obtain by exploiting graph structures.

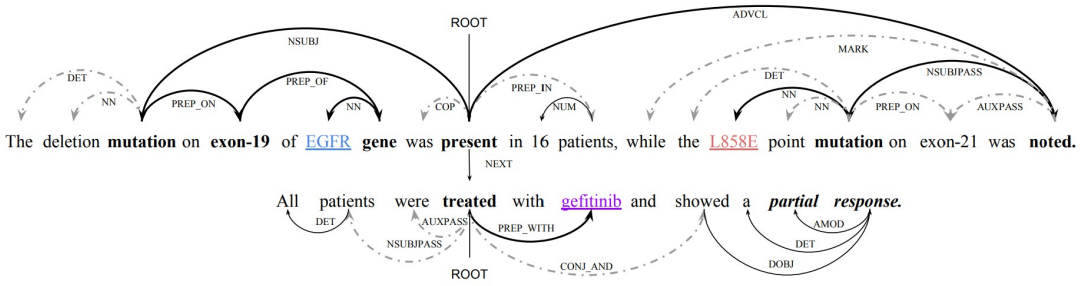


FIGURE 1.7: An example dependency graph for two sentences expressing a relation (*sensitivity*) among three entities: *L858E*, *EGFR* and *gefitinib*.

In order to remove irrelevant information associated with the dependency graph, various rule-based pruning strategies are also proposed to distill the dependency information in order to alleviate the error propagation. Xu et al. (2015b) and Xu et al. (2015c) apply neural networks

only on the shortest dependency path (SDP) between the entities in the full graph. Miwa and Bansal (2016) reduce the full tree to the subtree below the lowest common ancestor (LCA) of the entities. Zhang, Qi, and Manning (2018) use path-centric pruning, which only keeps tokens that are up to distance K away from the dependency path in the LCA subtree. Here K is a hyper-parameter. However, these static pruning strategies may exclude relevant information that is beneficial in predicting the correct relation. Figure 1.7 illustrates an example dependency graph corresponds to "*The deletion mutation on exon-19 of EGFR gene was present in 16 patients, while the L858E point mutation on exon-21 was noted. All patients were treated with gefitinib and showd a partial response.*" and how different pruning strategies are applied to the graph. The SDP between entities *EGFR*, *L858E* and *gefitinib* is highlighted in bold (tokens). The LCA subtree of entities is highlighted in bold (edges) and the root node is *present*. The dotted edges indicate tokens K=1 away from the subtree. However, the key tokens *partial response* that can be served as the cue for prediction would be excluded if the model only takes these pruned structures into consideration.

The error propagation becomes more severe in document-level and bio-medical relation extraction. For document-level texts, an accurate discourse parser is not available (Liu and Lapata, 2018). Prior efforts leverage a set of rules (Christopoulou, Miwa, and Ananiadou, 2019) or connect dependency trees of all sentences in the document (Sahu et al., 2019) to construct a document-level graph, which may not be able to capture the complex dependencies in a document. On the other hand, parsing accuracies drop significantly on the medical literature domain (McDonald et al., 2005). To address this issue, Song et al. (2019a) construct the dependency forest by using K-best dependency trees. Lifeng et al. (2020) learn the dependency parser and the relation extraction model jointly.

The major research challenge is how relation extraction models mitigate the error propagation and get the most out of the graph structure. Therefore, we ask the following two research questions:

- *Can we design a model that is able to learn how to prune the dependency graph from data?*
- *Can we induce a task-specific graph for relation extraction when an accurate parser is not available?*

We aim to design efficient graph-based models to solve these two research questions. A "soft pruning" strategy is first proposed to include relevant information and exclude irrelevant information associated with the dependency graph (Guo, Zhang, and Lu, 2019). For the second question, we treats the dependency graph as a latent variable and induce it in an end-to-end fashion (Nan et al., 2020; Guo et al., 2020)

1.3 Thesis Outline

The thesis is organized as follows:

- Chapter 2 presents the background of predicting graphs from texts (i.e. semantic parsing), and leveraging graphs for natural language generation and information extraction. We introduce the task definitions, the current approaches and the challenges in these tasks.

- Chapter 3 introduces a simple yet effective transition-based system for semantic graph (i.e. Abstract Meaning Representation) parsing. We argue that a well-defined search space for a transition system is crucial for building an effective parser. We propose to conduct the search in a refined search space based on a new compact AMR graph and an improved oracle. Our end-to-end parser achieves the state-of-the-art performance on various datasets with minimal additional information.

- Chapter 4 investigates the problem of encoding graphs for natural language generation. Unlike existing approaches where shallow graph convolutional networks (GCNs) were used, we introduce a dense connection strategy to GCNs and train a deeper GCN model for learning a better graph representation. Then we propose a dynamic fusion mechanism to capture richer non-local interactions by synthesizing higher order information from the input graphs. Two novel parameter saving strategies are further developed to reduce memory usage and model complexity.

- Chapter 5 explores how to leverage dependency structures for relation extraction. We propose a novel pruning strategy to improve the existing graph-based models. This strategy enables the model to maintain a better balance between including and excluding information associated with the dependency graph. We further develop latent structure induction approaches for document-level and bio-medical texts, on which accurate parsers are not available. Task-specific structures can be obtained in such a way to alleviate error propagation for better relation extraction.

- Chapter 6 concludes our efforts in predicting semantic graph from texts, and integrating graph structures for natural language generation and relation extraction. We summarize the overall research contribution in this thesis and what we would like to do in the future.

1.4 Contributions

Our overall contributions in this thesis can be summarized as follows:

- For better graph prediction, we present a novel transition-based system which is able to refine the search space (Guo and Lu, 2018). Extensive experiments show that our parser is able to achieve state-of-the-art performance with a simple architecture and minimal additional resources.

- In order to incorporate graph structure for language generation, we propose the novel densely connected graph convolutional networks (DCGCNs; Guo et al. 2019). Experimental results show that DCGCNs outperform existing models in two language generation tasks: AMR-to-text generation and syntax-based machine translation. Unlike previous designs of GCNs, DCGCNs scale naturally to significantly more layers without suffering from performance degradation and optimization difficulties.

- To better integrate high-order information, we propose the Lightweight, Dynamic Graph Convolutional Networks (LDGCNs) by introducing a novel dynamic fusion mechanism (Zhang et al., 2020). With the help of the mechanism, LDGCNs can effectively synthesize information from different orders to model complex interactions in graphs. We further develop two novel weight sharing strategies, which allow the LDGCN model to reduce memory usage and model complexity.

- To further improve the graph-based relation extraction model, we propose the novel AGGCNs that adopt a soft pruning strategy, which learns how to select and discard information associated with the dependency graph (Guo, Zhang, and Lu, 2019). AGGCNs achieves new state-of-the-art results on sentence-level relation extraction without additional computational overhead when compared with previous GCNs. Unlike tree-structured models (e.g., TreeLSTM; Tai, Socher, and Manning 2015), it can be efficiently applied over dependency trees in parallel.

- Parsing accuracies drop significantly on bio-medical and document-level texts, which leads to error propagation in downstream relation extraction tasks. In order to alleviate the error propagation, we treat the dependency structure as a latent variable and induce it in an end-to-end fashion. Task-specific structures can be obtained in this manner for better relational learning. Extensive experiments show that our proposed methods are able to achieve competitive results on document-level (Nan et al., 2020) and bio-medical (Guo et al., 2020) tasks without relying on external parsers.

Chapter 2

Background

This chapter introduces the background knowledge of predicting graphs from texts, and leveraging graphs for natural language generation and relation extraction. We present task definitions, existing approaches and research challenges in these three fields.

2.1 Predicting Semantic Graph

The task of predicting the corresponding semantic graph of an input sentence also known as semantic parsing. The semantic graph that this thesis works with is Abstract Meaning Representation (AMR; Banarescu et al. 2013). AMR is a general purpose meaning representation for the English language. It represents the meaning of a sentence as a direct, acyclic graph as shown in Figure 2.1 .

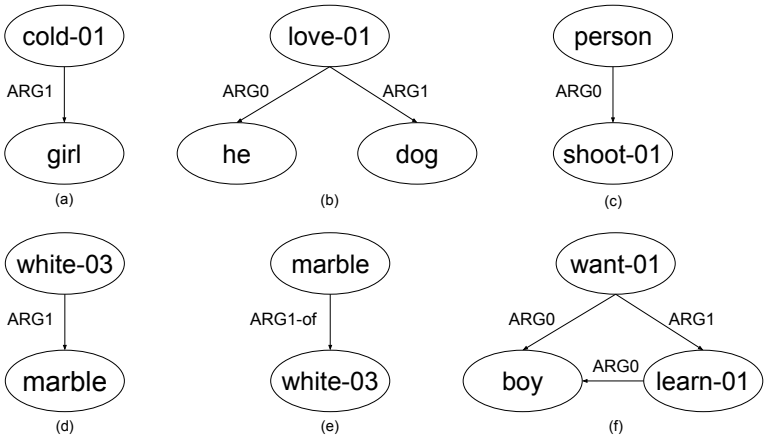


FIGURE 2.1: Example AMRs.

2.1.1 Abstract Meaning Representation

AMR focuses on sentence meaning, i.e. semantics. For example, the sentence *The girl is cold* is represented by the AMR in Figure 2.1 (a). The nodes in an AMR represent the concepts while the edges represent the relations between concepts. If a word is a noun, its corresponding node in the AMR is still the same. For example, the noun *girl* is the concept girl as shown in Figure 2.1 (a). If a word is a predicate, senses based on OntoNotes (Hovy et al., 2006) are used for disambiguation. For example, concept cold-01 means the temperature is low as used by the AMR in Figure 2.1 (a). Concept cold-02 means one is emotionless and unfriendly and cold-03

means something is no longer fresh. OntoNotes also provides argument roles for predicates, such as the ARG1 edge label here. This denotes that cold-01 has an ARG1 role, defined in OntoNotes as the thing being cold, and the girl has that role. This approach allows AMR to encode the who did what to whom of a sentence. Take the AMR for *He loves dogs* in Figure 2.1 (b) as an example. Here the concept he takes the ARG0 role of the lover while the concept dog take the ARG1 role as the object of affection.

In order to get a better understanding of AMR, we will introduce several important properties of AMR graphs. Firstly, AMR abstracts away from syntax. The sentences *He loves dogs*, *Dogs are loved by him* and *His love for dogs* all have the same AMR as shown in Figure 2.1 (b). Secondly, AMR represents no tense or aspect, e.g. the graph in Figure 2.1 (b) also represents the sentences *He loved dogs*, *He will love dogs*. In the same spirit, grammatical number and definiteness is dropped (*a dog*, *dogs*, *the dog* and *the dogs* are each just a dog concept), but explicit numbers are represented. Further, AMR is decompositional. For example, the AMR in Figure 2.1 (c), representing the word *shooter*. The AMR sometimes splits the meaning of a word into separate parts, allowing for more generalization. Here, *shooter* is represented as "a person who shoots". Further, each AMR has a root, which expresses the focus of the AMR. Usually this is the main predicate of the sentence. The difference becomes clearer when looking at AMRs for fragments without predicate: compare the root placement for *The marble is white* and *The white marble* as shown in Figures 2.1 (d) and (e) respectively. These graphs are also examples for how AMR displays modification.

In the examples we saw so far, all AMRs are tree structures. Actually, AMR can have a more complex structure with reentrancies as in the AMR for the sentence *The boy wants to learn* in Figure 2.1 (f). Here, the concept boy is both the wanter and the learner, and the additional ARG0 edge from learn-01 to boy results in an reentrancy.

AMR corpus contains real life sentences from both newspaper and online discussion forums, that go far beyond the complexity of the examples discussed here. AMR has been experimented with in some systems that take the graphs as input for an application, such as question answering (Mitra and Baral, 2016), text summarization (Liao, Lebanoff, and Liu, 2018) and machine translation (Song et al., 2019b).

2.1.2 AMR Parsing

AMR parsing, the task of transforming a sentence into its AMR graph as shown in Figure 2.2, is a challenging task as it requires the parser to learn to predict not only concepts, which consist of predicates, lemmas, named entities, wiki-links and co-references, but also a large number of relation types based on relatively sparse training data (Peng et al., 2017b). In this thesis, we categorize existing AMR parsers into two main classes: two-stage parser and one-stage parser.

There are also some exceptions staying beyond the above categorization. Peng, Song, and Gildea (2015) and Peng and Gildea (2016) develop a parser based on synchronous hyperedge replacement grammar while **artzi2015broad**; Misra and Artzi (2016) adapt combinatory categorical grammar. Pust et al. (2015) regard the task as a syntax-based machine translation. Groschwitz et al. (2018) and Lindemann, Groschwitz, and Koller (2019) leverage AM algebra to compose an AMR graph.

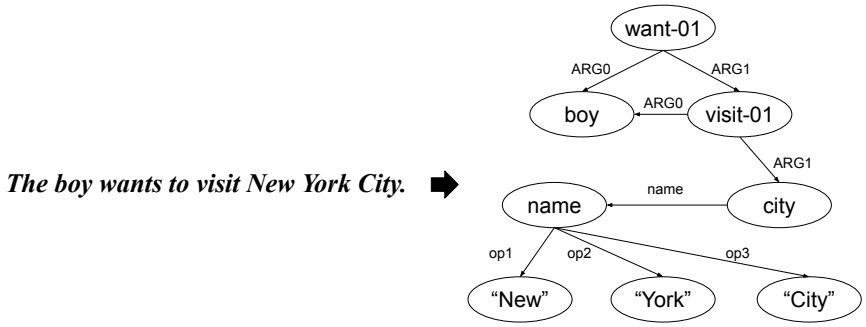


FIGURE 2.2: An example of parsing a sentence into its AMR.

2.1.3 Two-Stage Parser

Two-stage parsing adopts a pipeline design for concept identification and relation prediction. In the concept identification stage, the parser maps spans of words in the input sentence to concept graph fragments or to the empty graph fragment. For example, as shown in Figure 2.3, words in the sentence "*The boy wants to visit New York City*" is mapped to different graph fragments. ϕ here denotes empty fragments as stopwords (*The* and *to*) will not appear in the AMR graph. The span of words represents an entity ("*New York City*") is is mapped to a subgraph while other words is mapped to a single node.

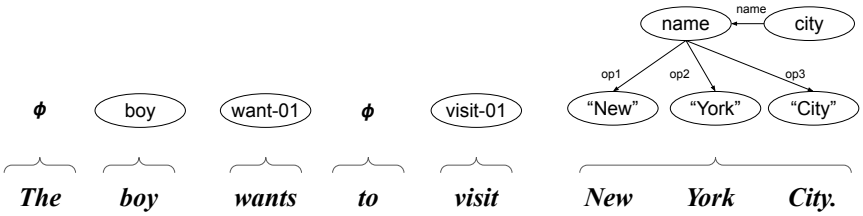


FIGURE 2.3: Concept Identification.

The relation prediction stage adds edges among the concept subgraph fragments identified in the first stage, creating a graph. As shown in Figure 2.4, edges and their labels will be attached in this stage. The concept want-01 becomes the root of the graph. A reentrancy is also created, concept boy has two parent concepts: want-01 and visit-01. The subgraph representing an entity is attached to the concept visit-01.

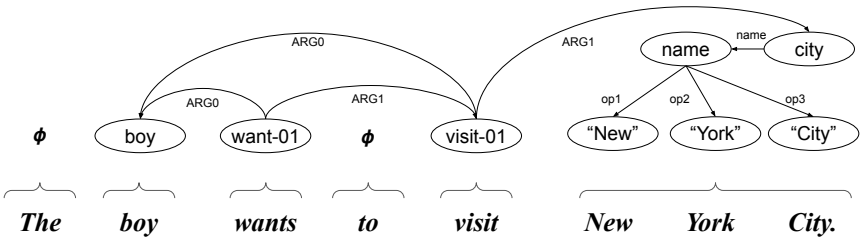


FIGURE 2.4: Relation Prediction.

Flanigan et al. (2014) propose the first two-stage AMR parser called JAMR. JAMR first finds the highest-scoring spans and concept graph fragments using a dynamic programming algorithm. Then a Maximum Spanning, Connected Subgraph Algorithm is used to adds edges to link these concepts into a fully connected AMR graph. Werling, Angeli, and Manning (2015) argue that the difficulty of AMR parsing lies in the first stage. Unlike JAMR, which uses a simple sequence model to directly generate AMR-subgraphs from memorized mappings of text spans to subgraphs, they make use of generative actions to generate these subgraphs. Flanigan et al. (2016a) improve the JAMR parser by using external resources and features and a novel loss function for structured prediction called infinite ramp. Recently, Foland and Martin (2017) and Lyu and Titov (2018) use bi-directional LSTM (Schuster and Paliwal, 1997) to discover the set of concepts and the set of predicate-argument relations among those concepts. On the other hand, Wang and Xue (2017) and Zhang et al. (2019a) also leverage recurrent neural networks to predict concepts in the first stage. In the second stage, Wang and Xue (2017) incorporate the predicted concepts into the CAMR parser (Wang, Xue, and Pradhan, 2015b). Zhang et al. (2019a) employ a deep bi-affine classifier (Dozat and Manning, 2017) to predict the relations. More recently, Zhou et al. (2020) further improve the concept identification stage by fusing gold and induced dependency structures.

As AMR annotation lacks explicit alignments between concepts in the graph and words in the sentences. An aligner is required to find the alignments between concepts and words before training a parser. Flanigan et al. (2014) design an aligner based on a set of rules, whereas Pourdamghani et al. (2014) develop an aligner based on statistical machine translation. Unlike previous work, Lyu and Titov (2018) treat the alignments as latent variables in a joint probabilistic model. Zhang et al. (2019a) extend the pointer-generator network (See, Liu, and Manning, 2017) for concept prediction. The pointer-generator network is based on the sequence-to-sequence (seq2seq) model (Sutskever, Vinyals, and Le, 2014; Bahdanau, Cho, and Bengio, 2015), which does not require any explicit alignments.

2.1.4 One-Stage Parser

One-stage parsers construct an AMR graph incrementally. For more fine-grained analysis, existing one-stage systems can be further categorized into three types: seq2seq-based, transition-based and graph-based methods.

Seq2seq-Based Parser Seq2seq-based systems train a modified seq2seq neural translation model with attention (Sutskever, Vinyals, and Le, 2014; Bahdanau, Cho, and Bengio, 2015) to translate plain English sentences into linearized AMRs, which are obtained by using depth-first search and both concepts and relations are treated as tokens (Barzdins and Gosko, 2016). Peng et al. (2017b) argue that seq2seq models mostly work well for large scale parallel data while the size of AMR corpus is much smaller. To overcome the data sparsity issue, they propose a categorization strategy which first maps low frequency concepts and entity subgraphs to a reduced set of category types. During decoding, they use the mapping dictionary constructed by heuristics to map the target types to their corresponding translation as a postprocessing procedure. Konstas et al. (2017) propose a paired training strategy to solve the data sparsity issue by using millions of unlabeled Gigaword sentences (Napoles, Gormley, and Durme, 2012). Similarly, Noord and Bos (2017) use 100k additional training pairs created by using CAMR (Wang, Xue, and Pradhan, 2015b) and JAMR(Flanigan et al., 2014). Unlike previous efforts, where linearized AMRs are used, Peng et al. (2018) propose a sequence-to-action-sequence approach

with a cache transition system, which provides better local context information than the linearized graph representation.

Transition-Based Parser Transition-based parsing processes a sentence from left-to-right and constructs the graph incrementally by alternately inserting a new node or building a new edge. Wang, Xue, and Pradhan (2015b) propose the first transition-based AMR parser called CAMR. CAMR first obtains the dependency trees of the input sentences, then greedily transduces the dependency tree into an AMR graph. To further improve the CAMR parser, Wang, Xue, and Pradhan (2015a) and Wang et al. (2016a) develop refined actions and incorporate additional linguistic features. Goodman, Vlachos, and Naradowsky (2016) use the DAGGER imitation learning algorithm (Ross, Gordon, and Bagnell, 2011) to generalize CAMR to unseen data. On the other hand, Damonte, Cohen, and Satta (2016) develop the first arc-eager transition-based parser that is able to directly parse an input sentence into its AMR graph. Buys and Blunsom (2017) use a graph-based variant of the arc-eager transition-system. The transition sequence, seen as a graph linearization, is predicted with the encoder-decoder architecture (Bahdanau, Cho, and Bengio, 2015). Ballesteros and Al-Onaizan (2017) apply stack LSTM (Dyer et al., 2015) to the newly designed transition system. Naseem et al. (2019) enrich the system by augmenting the training process with Policy Learning (Rennie et al., 2017) and rewarding the Smatch score (Cai and Knight, 2013) of sampled graphs. Peng, Gildea, and Satta (2018) apply the cache transition system to AMR parsing and design refined action phases, each modeled with a separate feed forward neural network. Vilares and Gómez-Rodríguez (2018) propose a non-projective parsing system that is able to better handle cycles and reentrancy in AMR graphs.

Unlike seq2seq-based methods, transition-based approaches require explicit alignments between concepts and words for training. Most of the previous parsers rely on the rule-based aligner (Flanigan et al., 2014). In order to build a better aligner, Wang and Xue (2017) propose a HMM-based aligner that integrates the graph distance distortion. Liu et al. (2018) propose a AMR aligner with rich semantic resources. They further propose a transition system for AMR parsing and use its oracle to tune the aligner by picking alignment that leads to the highest-scored achievable AMR graph.

Graph-Based Parser To alleviate the error propagation in the two-stage AMR parsers, Zhou et al. (2016a) propose the first one-stage graph-based parser, which performs concept identification and relation prediction simultaneously. They first develop a novel component wise beam search algorithm for relation prediction in an incremental fashion, and then incorporate the decoder into a unified framework based on multiple-beam search, which allows for the bi-directional information flow between the two subtasks in a single incremental model. Recent effort (Zhang et al., 2019b) propose an attention-based neural transducer that extends the two-stage AMR parser of Zhang et al. (2019a) to directly transduce input text into a meaning representation in one stage. This transducer builds a meaning representation incrementally. On the other hand, it uses multiple attention mechanisms to remove the need for aligners. Concurrently, Cai and Lam (2019) introduces Graph Spanning based Parsing (GSP), which constructs a parse graph incrementally in a top-down fashion. Starting from the root, at each step, a new node and its connections to existing nodes will be jointly predicted. The output graph spans the nodes by the distance to the root, following the intuition of first grasping the main ideas then digging into

more details. Cai and Lam (2020a) further improve the GSP by leveraging the mutual causalities between concept identification and relation prediction, and designing an iterative inference algorithm.

2.1.5 Challenges

The transition-based approach (Yamada and Matsumoto, 2003; Nivre, 2003) has been popular among many NLP tasks, including dependency parsing (Chen and Manning, 2014; Dyer et al., 2015), named entity recognition (Lample et al., 2016), and semantic parsing (Cheng et al., 2017). Different from the graph-based approach for structured prediction (e.g., conditional random fields; Lafferty, McCallum, and Pereira 2001), such an approach is able to maintain a good balance between efficiency and accuracy (Nivre, 2008), and has achieved state-of-the-art results on a number of tasks (Swayamdipta et al., 2016; Cheng et al., 2017; Shi, Huang, and Lee, 2017). While the transition-based approach is promising for AMR parsing, existing transition-based AMR parsers still cannot attain the state-of-the-art results on such a task. We observe that the key challenge in developing an effective transition-based system is a properly defined search space.

While for dependency parsing we only need to predict relation labels and the output vocabulary is limited to 37 (universal dependencies), AMR parsers need to predict both concepts and relation labels. At each time step, the transition-based parser is required to select one transition from tens of thousands of transitions.

The quality of alignments also affects the search space. Oracle, the algorithm used at the training time for specifying the action sequence that can recover the gold AMR graph, is also crucial for the transition-based system. A good oracle will be able to teach the parser how to find a proper path in the search space. The oracle requires alignment information between words and concepts.

2.2 Encoding Graph for Natural Language Generation

In this section, we will focus on graph-to-sequence learning, which can be framed as transducing graph structures to sequences for Natural Language Generation (NLG), such as AMR-to-text generation and syntax-based machine translation. Figure 2.5 shows an example of AMR-to-text generation, which is to recover the text representing the same meaning as an input AMR graph. The key challenge in these tasks is to efficiently learn effective graph representations.

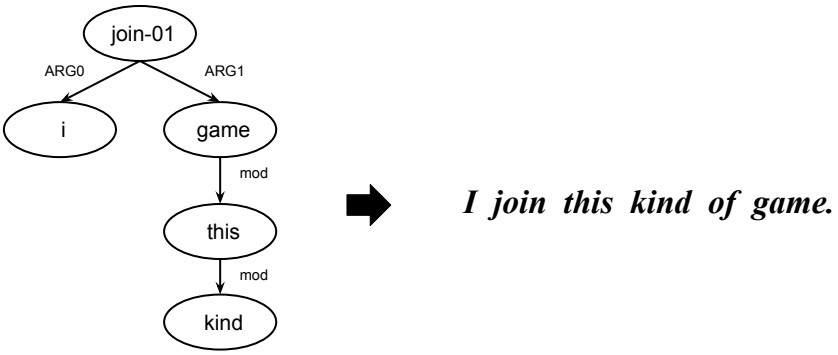


FIGURE 2.5: An example of AMR-to-text generation.

To capture structural information associated with graphs, we investigate the problem of encoding graphs using different neural encoders. Existing encoders can be categorized into three main types: sequence encoder, graph encoder and self-attention encoder.

2.2.1 Sequence Encoder

To encode graphs, early effort (Konstas et al., 2017) first linearize the input graph with depth-first traversal, before feeding it into a bi-directional LSTM (Schuster and Paliwal, 1997). However, linearizing the graph may lose crucial structural information. For example, originally closely-located graph nodes (such as parents and children) can be very far away, especially when the graph is very large. Graph has an important property that is invariance to ordering, which should ideally be reflected by a deep learning component for relational reasoning(Battaglia et al., 2018). However, different graph traversal methods will lead to different results sequences, which increases the data variation and further introduces ambiguity. Despite the drawbacks mentioned above, a sequence encoder is able to achieve competitive results given large-scale training data. For example, Konstas et al. (2017) leverage a sequence encoder to encode 20 million sentences paired with automatically parsed AMR graphs for text generation. The sequence-based model significantly outperforms previous statistical models (Pourdamghani, Knight, and Hermjakob, 2016; Flanigan et al., 2016a; Song et al., 2016; Song et al., 2017). These results show the power of neural networks for learning a good representation, but there still exists room for improvement due to the lost of structural information.

2.2.2 Graph Encoder

Early efforts that attempt to extend neural networks to deal with graphs are introduced by Gori, Monfardini, and Scarselli (2005) and Scarselli et al. (2009). More recent graph neural networks (GNNs) follow an information propagation scheme, where the representation of each node is iteratively updated based on those of its adjacent nodes in the graph. During an iteration, information is first aggregated for each node from its neighbors, then the information is applied to update the node state. The main difference between sequence encoder and graph encoder is that graph encoder do not require a node order for the input, such as a left-to-right order for sentences and a bottom-up order for trees (e.g. Tree-LSTM; Tai, Socher, and Manning 2015). In fact, GNNs only require the local neighborhood information, thus they are invariant to the permutations of the nodes and edges, which better captures the property of a graph.

For more fine-grained analysis, existing graph encoders can be further categorized into three main types: convolutional graph encoder, recurrent graph encoder and mixed encoder. The contrast between convolutional graph encoder and recurrent graph encoder is reminiscent of the contrast between Convolutional Neural Networks (CNNs; LeCun and Bengio 1995) and Recurrent Neural Networks (RNNs; Elman 1990).

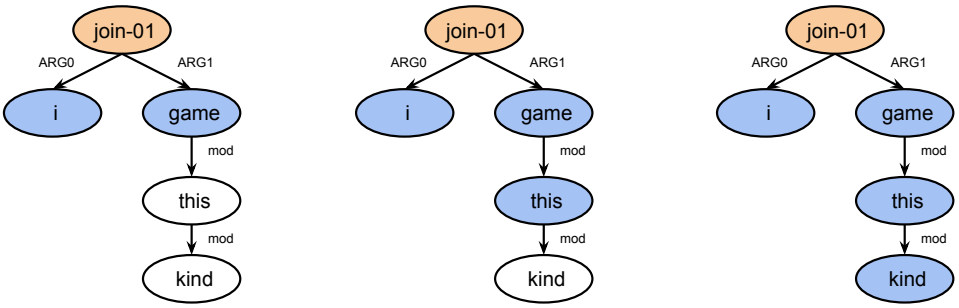
Convolutional Graph Encoder Bruna (2014) first apply the convolution operation on graph Laplacians to construct efficient architectures in the spectral domain. Subsequent efforts improve its computational efficiency with local spectral convolution techniques (Henaff, Bruna, and LeCun, 2015; Defferrard, Bresson, and Vandergheynst, 2016; Kipf and Welling, 2017). Existing convolutional graph encoders in NLP are closely related to Graph Convolutional Networks (GCNs; Kipf and Welling 2017), which restrict the filters to operate on a first-order neighborhood around each node.

Formally, given an AMR graph $\mathcal{G}$ with n concepts (nodes), GCNs associate each concept v with a feature vector $\mathbf{h}_v \in \mathbb{R}^d$, where d is the feature dimension. $\mathcal{G}$ can be represented by concatenating features of all the concepts, i.e., $\mathbf{H}$=$[\mathbf{h}_{v_1}, ..., \mathbf{h}_{v_n}]$. Graph convolutions at l-th layer can be defined as:

$$\mathbf{H}_{l+1} = \phi(\mathbf{A}\mathbf{H}_l\mathbf{W}_l + \mathbf{b}_l) \tag{2.1}$$

where $\mathbf{H}_l$ is hidden representations of the l-th layer. $\mathbf{W}_l$ and $\mathbf{b}_l$ are trainable model parameters for the l-th layer, ϕ is an activation function. $\mathbf{A}$ is the adjacency matrix, $\mathbf{A}_{uv}$=1 if there exists a relation (edge) that goes from concept u to concept v.

The first layer of GCNs can only capture the graph's adjacency information between immediate neighbors, while with the second layer one will be able to capture second-order proximity information (neighborhood information two hops away from one node). As shown in Figure 2.6, concept join-01 collect information from immediate neighbor concepts i and game in the first graph convolutional layer. In the third layer, concept join-01 is able to collect information from the third-order neighbor concept kind. L layers will be needed in order to capture neighborhood information that is L hops away.



(a) First GCN Layer (b) Second GCN Layer (c) Third GCN Layer

FIGURE 2.6: Each graph convolutional layer encodes information from immediate neighbors. The third layer can capture third-order neighborhood information (nodes that are 3 hops away from the current node join-01).

Recent improvements and extensions of GCNs include using additional aggregation methods such as vertex attention (Veličković et al., 2018) or gating mechanism (Hamilton, Ying, and Leskovec, 2017) to better summarize neighborhood states. Although deeper GCNs with more layers will be able to capture richer neighborhood information of a graph, empirically it has been observed that the best performance is achieved with a 2-layer model (Li, Han, and Wu, 2018; Xu et al., 2018). For the graph-to-sequence learning task, a shallow convolutional architecture may not be able to capture the complex interactions for large graphs.

Recurrent Graph Encoder Therefore, recent efforts that leverage recurrence based graph neural networks have been explored as the alternatives to encode the structural information of graphs. Deep architectures based on such recurrent models have been successfully built for natural language generation, where rich neighborhood information captured was shown useful. Specifically, gated graph neural networks (GGNNs; Li et al. 2016b; Beck, Haffari, and Cohn 2018) leverage the gating operation of Gated Recurrent Units (GRUs; Cho et al. 2014 to update the representation of nodes. The gating operation helps to learn long distance relations between nodes and avoid gradient diminishing in the recurrent process.

$$\mathbf{R}_{l+1} = \sigma(\mathbf{A}\mathbf{H}_l\mathbf{W}_l^r + \mathbf{b}_l^r) \tag{2.2}$$
$$\mathbf{Z}_{l+1} = \sigma(\mathbf{A}\mathbf{H}_l\mathbf{W}_l^z + \mathbf{b}_l^z) \tag{2.3}$$
$$\tilde{\mathbf{H}}_{l+1} = \rho(\mathbf{A}(\mathbf{R}_{l+1} \odot \mathbf{H}_l)\mathbf{W}_l + \mathbf{b}_l) \tag{2.4}$$
$$\mathbf{H}_{l+1} = (1 - \mathbf{Z}_{l+1}) \odot \mathbf{H}_l + \mathbf{Z}_{l+1} \odot \tilde{\mathbf{H}}_{l+1} \tag{2.5}$$

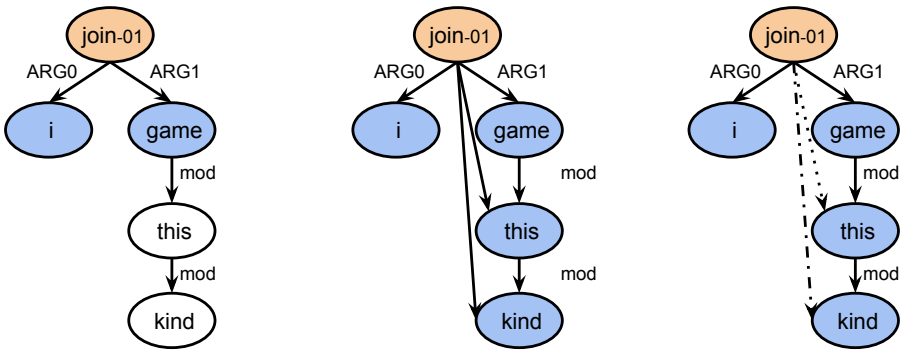
where $\mathbf{H}_l$ is hidden representations of the l-th layer. $\mathbf{W}_l^r$, $\mathbf{W}_l^z$, $\mathbf{W}_l$, $\mathbf{b}_l^r$, $\mathbf{b}_l^z$ and $\mathbf{b}_l$ are trainable model parameters for the l-th layer, σ is the sigmoid function and ρ is an activation function. $\mathbf{A}$ is the adjacency matrix, $\mathbf{A}_{uv}$=1 if there exists a relation (edge) that goes from concept u to concept v. Using the above gating mechanism, information from each node is able to propagate to all its neighboring nodes.

Similarly, Song et al. (2018a) and Zhang, Liu, and Song (2018) propose graph recurrent networks (GRNs), which applies the gating mechanism of LSTM to perform the update.

Mixed Graph Encoder Another solution is stacking the structural information provided by a convolutional graph encoder with the sequential information obtained from a sequence encoder. This has been shown to be effective for other tasks with GCNs (Marcheggiani and Titov, 2017; Zhang, Qi, and Manning, 2018). Bastings et al. (2017) and Damonte and Cohen (2019) first pass the input to a sequence encoder (e.g. CNNs or RNNs), the output of which is then fed into the convolutional graph encoder. Damonte and Cohen (2019) also propose an alternative solution where the sequence encoder is used on top of the convolutional graph encoder. The input representations are refined by exploiting the explicit structural information given by the graph. The refined representations are then fed into the sequence encoder to model the long-range dependencies.

2.2.3 Self-Attention Encoder

Unlike the GNNs, self-attention networks (SANs; Vaswani et al. 2017) capture global interactions by connecting each concept to all other concepts as shown in Figure 2.7 (b). Intuitively, the attention matrix can be treated as the adjacency matrix of a fully-connected graph.



(a) Vanilla GCNs (b) SANs (c) Structured SANs

FIGURE 2.7: Comparisons between convolutional graph encoder and self-attention encoder.

Formally, SANs take a sequence of representations of n nodes $\mathbf{H}=[\mathbf{h}_{v_1}, ..., \mathbf{h}_{v_n}]$ as the input. The attention score $\mathbf{A}_{uv}$ between the concept pair (u,v) can be defined as:

$$\begin{aligned}\mathbf{A}_{uv} &= f(\mathbf{h}_u, \mathbf{h}_v) \\ &= \text{softmax}(\frac{(\mathbf{h}_u\mathbf{W}^Q)(\mathbf{h}_v\mathbf{W}^K)^T}{\sqrt{d}})\end{aligned} \tag{2.6}$$

where $\mathbf{W}^Q$ and $\mathbf{W}^K$ are projection parameters. The adjacency matrix $\mathbf{A}$ in GCNs is given by the input AMR graph, while in SANs $\mathbf{A}$ is computed based on $\mathbf{H}$, which neglects the structural information of the input AMR. The number of operations required by graph convolutions scales is found linearly in the input length, whereas they are quadratic for SANs.

Structured Self-Attention Encoder Zhu et al. (2019) and Cai and Lam (2020b) extend the self-attention encoder by incorporating the relation $\mathbf{r}_{uv}$ between node u and node v in the original graph such that the model is aware of the input structure when computing attention scores:

$$\mathbf{A}_{uv} = g(\mathbf{h}_u, \mathbf{h}_v, \mathbf{r}_{uv}) \tag{2.7}$$

where $\mathbf{r}_{uv}$ is obtained based on the shortest relation path between the concept pair (u, v) in the graph. For example, the shortest relation path between the concept pair (join-01, this) in Figure 2.7 (c) is [ARG1, mod]. Formally, the path between concept u and v is represented as s_{uv}=$[e(u, k_1), e(k_1, k_2), ..., e(k_m, v)]$, where e indicates the relation label between two concepts and $k_{1:m}$ are the relay nodes.

The concept join-01 in vanilla GCNs only captures information from its immediate neighbors (first-order). In SANs, the node collects information from all other nodes, while in structured SANs it is aware of its connected nodes in the original graph.

2.2.4 Challenges

Compared with recurrent neural networks, convolutional architectures are highly parallelizable and are more amenable to hardware acceleration (Gehring et al., 2017). On the other hand, convolutional operations are more computationally efficient than self-attention operations because the computation of attention weights scales quadratically while convolutions scale linearly with respect to the input length (Wu et al., 2019a). Therefore, it is worthwhile to explore the possibility of encoders based on graph convolutions. However, there exist two main challenges when building an expressive convolutional graph encoder.

The first challenge is how to train a deeper GCN for modeling large and complex graphs. Prior efforts have tried to train deep GCNs by incorporating residual connections (Bastings et al., 2017). Xu et al. (2018) show that vanilla residual connections proposed by He et al. (2016) are not effective for graph neural networks. They next attempt to resolve this issue by adding additional recurrent layers on top of graph convolutional layers. However, they are still confined to relatively shallow GCNs architectures (at most 6 layers in their experiments), which may not be able to capture the complex interactions for larger graphs.

The second challenge is how to capture non-local information in graph convolutional operations. Previous works (Zhu et al., 2019; Cai and Lam, 2020b; Wang, Wan, and Jin, 2020) have shown that the locality property of existing GCNs precludes efficient non-local information propagation. Abu-El-Haija et al. (2019) further prove that vanilla GCNs are unable to

capture feature differences among neighbors from different orders no matter how many layers are stacked.

2.3 Modelling Dependency Graph for Relation Extraction

Apart from NLG, relation extraction (RE) models also benefit from incorporating structural information, such as the dependency graph (Zhang, Qi, and Manning, 2018). Relation extraction aims to detect relations among entities in the text. It plays a significant role in a variety of natural language processing applications including biomedical knowledge discovery (Quirk and Poon, 2017), knowledge base population (Zhang et al., 2017b) and question answering (Yu et al., 2017). For fine-grained analysis, we categorize the relation extraction task into two levels: sentence-level and document-level.

Dependency parsing has achieved an accuracy over 96% in the news domain (Zhang, Li, and Zhang, 2020). However, for other domains, parsing accuracies can drop significantly. This can lead to severe error propagation in downstream relation extraction tasks, offsetting much of the benefit that relation extraction models can obtain by exploiting dependency graphs as a source of external features. Therefore, we also consider the out-of-domain setting. Specifically, we focus on the bio-medical relation extraction.

2.3.1 Sentence-Level Relation Extraction

Sentence-level relation extraction involves discerning whether a relation exists between two entities in a sentence (often termed subject and object, respectively). Figure 2.8 shows an example from the TACRED dataset (Zhang et al., 2017b). The relation between the subject entity *Cathleen P. Black* and the object entity *chairwoman* is title, and per denotes the relation category belongs to person.

> Carey will succeed **Cathleen P. Black**, who held the position for 15 years and
> will take on a new role as **chairwoman** of Hearst Magazines, the company said.
>
> **Subject:** *Cathleen P. Black* **Object**: *chairwoman* **Relation**: *per:title*

FIGURE 2.8: An example from the TACRED corpus.

Most existing sentence-level relation extraction models can be categorized into two classes: sequence-based and graph-based. Sequence-based models operate only on the word sequences (Zeng et al., 2014; Wang et al., 2016b), whereas graph-based models incorporate dependency graphs into the models (Bunescu and Mooney, 2005; Peng et al., 2017a). Compared to sequence-based models, graph-based models are able to capture non-local dependency relations that are obscure from the surface form alone (Zhang, Qi, and Manning, 2018). In this thesis, we mainly focus on how to model dependency graphs in relation extraction.

Graph-Based Model Early research efforts are based on statistical methods. Tree-based kernels (Zelenko, Aone, and Richardella, 2002) and dependency path-based kernels (Bunescu and

Mooney, 2005) are explored to extract the relation. McDonald et al. (2005) construct maximal cliques of entities to predict relations. Mintz et al. (2009) include syntactic features to a statistical classifier.

Recently, graph-based approaches also try to incorporate structural information into the neural models. Peng et al. (2017a) first split the dependency graph into two Directed Acyclic Graphs (DAGs), then extend the tree LSTM model (Tai, Socher, and Manning, 2015) over these two graphs for n-ary relation extraction. An obvious drawback is that structural information is lost by splitting a graph into two DAGs. To address this issue, Linfeng et al. (2018) use GRNs (Song et al., 2018a) to directly encode the whole dependency graph without breaking it.

Pruning Strategy However, incorporating a dependency graph from an external parser may introduce noises into the model. To maintain a balance between including and excluding information associated with the dependency graph, various pruning strategies are also proposed to distill the structural information. Xu et al. (2015b) and Xu et al. (2015a) apply neural networks only on the shortest dependency path between the entities in the full tree. Liu et al. (2015) combine the shortest dependency path and the dependency subtree. Miwa and Bansal (2016) reduce the full tree to the subtree below the lowest common ancestor (LCA) of the entities. Zhang, Qi, and Manning (2018) apply GCNs (Kipf and Welling, 2017) model over a pruned tree. This tree includes tokens that are up to distance K away from the dependency path in the LCA subtree.

2.3.2 Document-Level Relation Extraction

A more challenging, yet practical extension, is the document-level relation extraction, where a system needs to comprehend multiple sentences to infer the relations among entities by synthesizing relevant information from the entire document (Jia, Wong, and Poon, 2019; Yao et al., 2019). Figure 2.9 shows an example adapted from the recently proposed document-level dataset DocRED (Yao et al., 2019). The example has four entities: *Lutsenko*, *internal affairs*, *Yulia Tymoshenko* and *Ukrainian*. Here entity *Lutsenko* has two mentions: *Lutsenko* and *He*. Mentions corresponding to the same entity are highlighted with the same color. The solid and dotted lines represent intra- and inter-sentence relations, respectively.

In order to infer the inter-sentence relation (i.e., country of citizenship) between *Yulia Tymoshenko* and *Ukrainian*, one first has to identify the fact that *Lutsenko* works with *Yulia Tymoshenko*. Next we identify that *Lutsenko* manages *internal affairs*, which is a *Ukrainian* authority. After incrementally connecting the evidence in the document and performing the step-by-step reasoning, we are able to infer that *Yulia Tymoshenko* is also a *Ukrainian*.

Graph Construction Document-level relation extraction requires integrating information within and across multiple sentences of a document and capturing complex interactions between inter-sentence entities. Prior efforts (Verga, Strubell, and McCallum, 2018; Jia, Wong, and Poon, 2019) leverage Multi-Instance Learning (MIL; Riedel, Yao, and McCallum 2010; Surdeanu et al. 2012) to model the interactions between mentions of entities.

On the other hand, structural information has been used to perform better reasoning since it models the non-local dependencies that are obscure from the surface form alone. Sahu et al. (2019) build a document-level graph, whose nodes correspond to words and edges represent local and non-local dependencies among them. The document-level graph is formed by connecting words with local dependencies from syntactic parsing and sequential information, as

FIGURE 2.9: An example from the DocRED corpus.

well as non-local dependencies from co-references and dependencies (Peng et al., 2017a). Relations between entities are inferred using MIL-based bi-affine pairwise scoring function (Verga, Strubell, and McCallum, 2018) on the entity node representations.

Instead of using external parsers, Christopoulou, Miwa, and Ananiadou (2019) build a document-level graph based on rules. In the proposed graph, a node corresponds to either entities, mentions, or sentences, instead of words. Nodes are connected based on simple heuristic rules and different edge representations are generated for the connected nodes. An edge-oriented model (Christopoulou, Miwa, and Ananiadou, 2018) is developed to learn edge representations for relational learning. An iterative algorithm over the graph edges is used to model dependencies between the nodes in the form of edge representations. The intra- and inter-sentence entity relations are predicted by employing these edges.

2.3.3 Bio-Medical Relation Extraction

The deletion mutation on exon-19 of **EGFR** gene was present in 16 patients, while the **L858E** point mutation on exon-21 was noted in 10. All patients were treated with **gefitinib** and showed a partial response.

Gene: *EGFR* **Mutation Point**: *L858E* **Drug**: *gefitinib*
Relation: *sensitive*

FIGURE 2.10: An example from the PubMed corpus.

The sheer amount of bio-medical articles and their rapid growth prevent scientists from receiving comprehensive knowledge by direct reading. This may hamper both bio-medical research and clinical diagnosis. NLP techniques have been used for automating the knowledge extraction process from the medical literature. Among these techniques, relation extraction plays a significant role as it facilitates the detection of relations among entities in the literature. Consider the following example from the PubMed dataset (Peng et al., 2017a) as shown in Figure 2.10. Collectively, these two sentences convey the fact that there is a ternary interaction between the three entities (gene, mutation point and drug), which is not expressed in either sentence alone. Namely, tumors with *L858E* mutation in *EGFR* gene can be treated with *gefitinib*. Extracting such knowledge clearly requires moving beyond binary entities and single sentences.

Error Propagation In the medical domain, early efforts leverage graph LSTM (Peng et al., 2017a) or graph neural networks (Song et al., 2018a; Guo, Zhang, and Lu, 2019) to encode the 1-best dependency tree. However, dependency parsing accuracy is relatively low in the medical domain. Figure 2.11 (a) shows the 1-best dependency tree obtained by the Stanford CoreNLP (Manning et al., 2014), where the dependency tree contains an error. In particular, phrase *phenylalanine hydroxylase* and *catecho* are gene and drug entity, respectively. The entity phrase *phenylalanine hydroxylase* is broken since the word *hydroxylase* is mistakenly considered as the main verb.

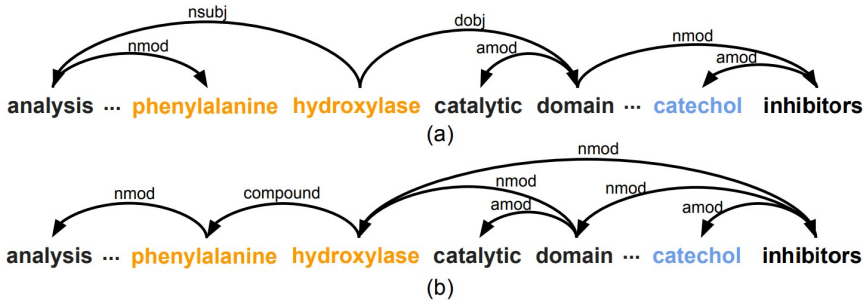


FIGURE 2.11: (a) 1-best dependency tree; (b) manually labeled gold tree.

In order to mitigate the error cascading caused by an out-of-domain parser, Song et al. (2019a) construct a dependency forest by adding additional edges with high confidence scores given by a dependency parser trained on the news domain (Dozat and Manning, 2017) or merging the K-best trees (Eisner, 1996) by combining identical dependency edges. Lifeng et al. (2020) jointly train a pre-trained dependency parser (Dozat and Manning, 2017) and a relation extraction model. The dependency forest generated by the parser is a 3-dimensional tensor, with each position representing the conditional probability of one word modifying another word with a relation, which encodes all possible dependency relations with the confidence scores.

2.3.4 Challenges

There exist two main challenges in building the graph-based relation extraction model. For the sentence-level relation extraction task, the main issue is how to maintain a better balance between keeping and removing information associated with the dependency graph. Existing rule-based pruning strategies might eliminate some important information in the full dependency graph. Ideally, the model should be able to learn how to include and exclude information in the full dependency graph.

The second challenge lies in document-level and bio-medical relation extraction is the lack of accurate dependency graph. For document-level relation extraction, sentence-level dependency trees, co-references and heuristics are used to build a document-level graph for relational reasoning since an accurate discourse parser is not available. However, such a document-level graph may not be able to capture the complex interactions among mentions and entities, especially when they appear in different sentences. For bio-medical relation extraction, previous efforts rely on dependency parsers trained on newswire text. The performance of the external parser drops significantly on the bio-medical domain. Therefore, parsing errors will downstream the pipeline due to the inaccurate dependency structure.

Chapter 3

Predicting Semantic Graph with a Refined Search Space

3.1 Introduction

In this chapter, we mainly focus on the transition-based AMR parsing system. As discussed in Section 2.1.5, the key challenge in developing an effective transition-based AMR parser is a properly defined search space. We argue that the search space used in existing transition systems needs to be refined. To this end, we design a new compact AMR graph representation. Transition actions designed based on such a compact graph enable our parser to generate the target structure with fewer actions and to better capture the correspondence between concepts and tokens in the sentence.

Oracle, the algorithm used at the training time for specifying the action sequence that can recover the gold AMR graph, is also crucial for the transition-based system. A good oracle will be able to teach the parser how to find a proper path in the search space. The oracle requires alignment information between words and concepts. We identify limitations associated with current practice for finding such alignment information, and propose a new approach that integrates both rules and unsupervised learning.

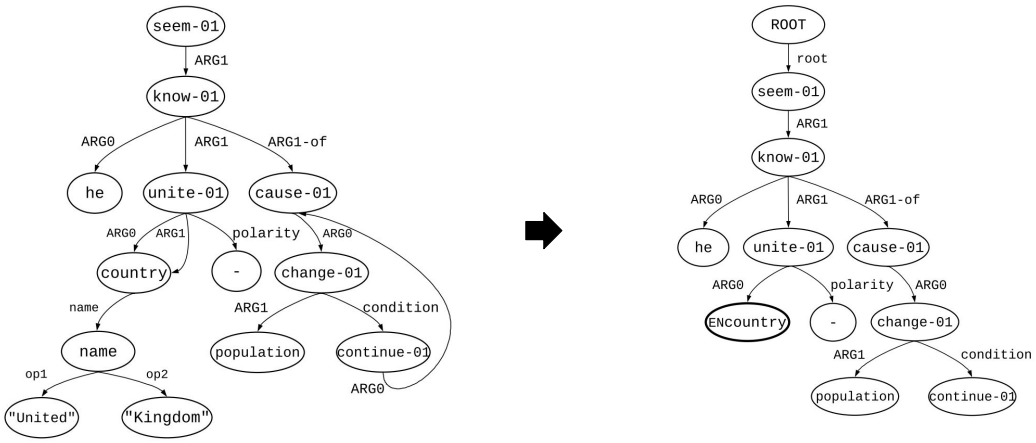
Experiments show that our system that makes use of POS tags as the only external resource performs competitively on benchmark datasets. To the best of our knowledge, the parser achieves the highest score on the standard LDC2014T12 dataset. Our parser also yields competitive scores on the LDC2015E86 dataset and the more recent LDC2017T10 dataset. On the popular newswire section of LDC2014T12 dataset, our parser outperforms the previous best system by around 3 points in terms of F_1 measure.

3.2 Approach

Firstly, we propose a new compact representation for an AMR graph. Based on our new representation, we further present a novel technique for constructing the action sequence used for training our model. As we will see later, both newly introduced techniques are crucial for building an improved transition-based system within our refined search space.

3.2.1 Compact AMR Graph

As shown in Figure 3.1, we design a representation called *compact AMR graph* to simplify concepts and relations of an AMR graph, which makes the learning of our transition system



(a) AMR Graph

(b) Compact AMR Graph

FIGURE 3.1: An AMR (a) and its compact graph (b). They correspond to the sentence: *It seems that he knows under the population changes (if it continues) that U.K. will not unite.*

easier. The construction of our compact AMR graph involves removing concepts and relations from an original AMR graph.

Remove Concepts First we categorize AMR concepts into 2 types:

- **Lexical**: concepts which are converted directly from tokens in the sentence into certain expressions ranging from predicates with sense tags, lemmas to tokens with quotation marks. One example is the concept seem-01 shown in Figure 3.1 (a), which is converted from the token *seems* with 01 as the sense tag.
- **Non-Lexical**: concepts which are invoked by their child concepts rather than from tokens in the sentence directly. Examples include country and name in Figure 3.1 (a).[1]

A non-lexical concept is invoked by its child concepts, while a lexical concept corresponds to a certain token in the input sentence. Inspired by Lu et al. (2008) which learns a semantic parser by aligning words and semantic units, we compress a subgraph rooted by a non-lexical concept into a new concept, which directly corresponds to a contiguous span in the sentence.

For example, a subgraph that consists of non-lexical concepts: country, name and lexical concepts "United", "Kingdom" in Figure 3.1 (a) can be compressed into one non-lexical concept ENcountry, which can be aligned to the token *"U.K."* in the input sentence as shown in Figure 3.1 (b).

Removing concepts in the graph enables the parser to build it with fewer actions. Empirically we find that 9% fewer actions are required in LDC2015E86 on average. Also, all concepts can be invoked by a contiguous span in the sentence, which helps the parser to capture the correspondence between concepts and tokens better.

[1] In practice, we define non-lexical concepts based on rules.

Remove Relations We also refine the search space by eliminating certain relations in the original AMR graph. The number of relation types is relatively large in the AMR corpus, which leads to a large search space.

One of the main properties of AMR, and the reason why sentences are represented as graphs rather than trees, is the presence of nodes with multiple parents, called reentrancies (Szubert et al., 2020). For example, the concept country shown in Figure 3.1 (a).

In order to introduce *reentrancy*[2], attached concepts have to stay in the stack rather than being removed as in most of transition-based AMR parsers.

Take cause-01 shown in the Figure 3.1 (a) as an example, which is a reentrancy node headed by know-01 and continue-01. After it has been attached to its parent know-01 and child change-01, it still needs to stay in the stack and wait for another relation headed by continue-01. These two concepts are potentially far away from each other in the input sentence, which means cause-01 has to stay in the stack for a long time. In general, the longer a word has to wait to get assigned the more opportunities there are for something to go awry (Manning and Schütze, 1999).

During the testing phase, the parser does not know whether cause-01 is a reentrancy node. Therefore, the parser needs to add all possible relation attachment actions (nearly half of the action space) into the valid action set as long as cause-01 exists in the stack, which makes the parser harder to train.

Thus, we define several properties that the compact graph should respect to further refine the search space[3]:

- **Acyclicity**: cycles are forbidden. The relation ARG0 between cause-01 and continue-01 shown in Figure 3.1 (a) is removed in the compact AMR graph Figure 3.1 (b).
- **Simple**: for each parent-child concept pair, only one relation is attached. There exist two parallel relations ARG0 and ARG1 between concepts unite and country shown in Figure 3.1 (a). One of them is removed in Figure 3.1 (b).
- **Non-terminal restricted**: only a subset of concepts are allowed to have children. For example, lexical concept **–** in Figure 3.1 (a) can only be a terminal node in the compact AMR graph, which means that it can be removed from the stack once it has been attached.
- **Reentrancy restricted**: reentrancy is forbidden for certain concepts, which means these concepts only have one parent concept. An example is that lexical concepts **–** cannot be reentrancy in the compact AMR graph.

After incorporating these constraints, relation attachment actions are forbidden at many states, which refines the search space. Even though such constraints might prevent us from generating some valid AMR graphs, in practice we can convert the compact AMR graph back to the AMR graph without much loss[4].

[2] One concept can participate in multiple relations

[3] In practice, we impose constraints on the valid action set of the parser for each state to ensure the growing structure always respects these properties.

[4] 90% of graphs in the training set of LDC2015E86 dataset satisfy these constraints, which means no relation needs to be removed for most of the graphs.

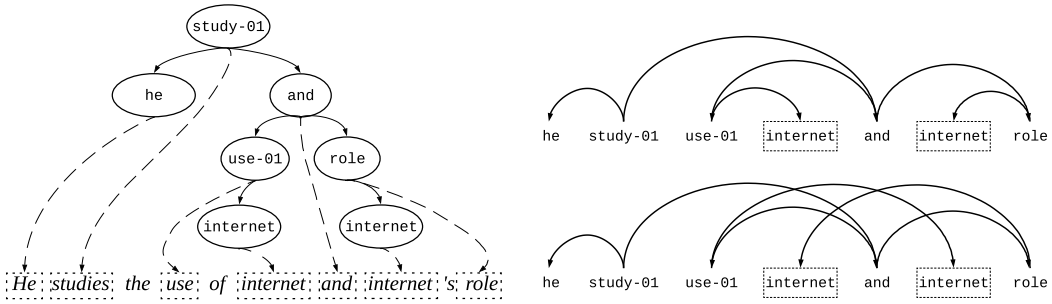


FIGURE 3.2: An alignment example for a sentence that has two identical tokens *internet* and the relation arcs between concepts if we put them on a semi-plane.

3.2.2 Oracle

Another key to successful search is the oracle, an algorithm that produces a sequence of actions that lead to the gold AMR graph. The action sequence generated by the oracle is significant as it tells the model which actions should be taken during training. The oracle requires, as input, the alignment between tokens in the sentences and concepts in the graph. Almost all prior transition-based parsers have used the JAMR aligner for identifying such alignment information. However, we find that the JAMR aligner suffers from two issues, which may lead to errors that may be propagated to the parser through the oracle.

Non-Projectivity Caused by Alignment Error Consider a sentence that has two identical tokens–*internet*. Figure 3.2 illustrates the alignments between concepts and tokens. Since our parser predicts the concepts from left to right, we can draw edges between aligned concepts on a semi-plane to verify whether it has crossing arcs. If the alignment is correct, the AMR graph is projective as shown at the top right of Figure 3.2. However, the JAMR is a rule-based aligner, which uses a set of heuristics to do fuzzy matching. Its matching mechanism tends to make mistakes when there exist multiple identical words. Such alignment errors may lead to crossing arcs that our parser may not be able to handle.

Assume an alignment error occurs as illustrated in the bottom right of Figure 3.2. Such an error will force the parser to attach relations between wrong concepts. Sometimes these concepts are far from each other, which potentially leads to more crossing arcs. For example, ARG1 is attached to the wrong internet, causing the undesired crossing arc.

Empty Alignments Another issue is that as the corpus gets larger, the JAMR aligner yields more empty alignments as shown in Table 3.3. Since the set of matching rules is static, unseen patterns in a larger corpus may result in sub-optimal alignment information.

Hybrid Aligner In order to address these issues, we first try the ISI aligner (Pourdamghani et al., 2014). According to preliminary results (Table 3.3), we find that the performance of such an unsupervised model is not as good as the JAMR aligner when aligning relation and non-lexical concepts. Therefore, we propose a hybrid aligner, which combines unsupervised learning and rule-based strategy.

Action	$State_t \rightarrow State_{t+1}$
SHIFT	$(S, u\|B) \rightarrow (u\|S, B)$
REDUCE	$(u\|S, B) \rightarrow (S, B)$
RIGHTLABEL(r)	$(u\|S, v\|B) \rightarrow (g_r(u, v, r)\|S, v\|B)$
LEFTLABEL(r)	$(u\|S, v\|B) \rightarrow (u\|S, g_r(v, u, r)\|B)$
SWAP	$(u, v\|S, B) \rightarrow (v, u\|S, B)$
MERGE	$(u\|S, v\|B) \rightarrow (S, g_m(u, v)\|B)$
PRED(n)	$(S, u\|B) \rightarrow (S, n\|B)$
ENTITY(ℓ)	$(S, u\|B) \rightarrow (S, g_\ell(u, \ell)\|B)$
GEN(n)	$(S, u\|B) \rightarrow (S, u, n\|B)$

TABLE 3.1: Definition of actions. $(u|S)/(u, v|S)$: item u (or u, v) is at the top of the stack. g_r, g_m and g_ℓ represent composition functions.

JAMR does not consider information about the structure whereas the unsupervised models can capture *locality* (Wang and Xue, 2017) – the assumption that words that are adjacent in the source sentence tend to align to words that are closer in the target sentence (Vogel, Ney, and Tillmann, 1996). Structural information can also be incorporated into the model to allow it to capture locality beyond linearity (Wang and Xue, 2017). This property of the unsupervised aligner can alleviate the problem of non-projectivity caused by alignment error. Similar to what is done in the JAMR aligner, we also design rules based on properties of AMR graphs to improve alignments of non-lexical concepts.

In the preprocessing stage of the hybrid aligner, we remove all relations. As non-lexical concepts can be aligned to their child concepts in the compact graph, we then remove all non-lexical concepts. Our unsupervised method is based on IBM word alignment models (Brown et al., 1993). In the postprocessing stage, we align non-lexical concepts iteratively to the same span that its child concepts are aligned to. For example, non-lexical concepts country and name shown in Figure 3.1 (a) are removed in preprocessing. During postprocessing, their alignments come from child concepts "United" and "Kingdom", they are aligned to the 16-th token *"United"* and the 17-th token *"Kingdom"* respectively. Therefore, non-lexical concepts country and name can be aligned to the span 16-17, which is *"United Kingdom"*.

3.2.3 Transition System

The transition system consists of a stack S containing words that have been processed, a buffer B containing words to be processed. Initially, S_1 is empty and B_1 contains the whole input sentence and a end-of-sentence symbol at the end. Execution ends on time step t such that B_t is empty and S_t contains a single structure.

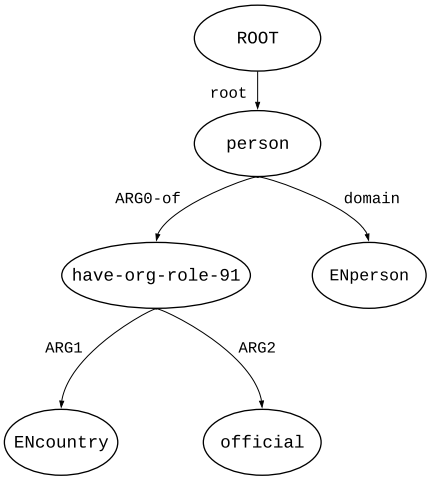
Motivated by (Henderson et al., 2013; Ballesteros and Al-Onaizan, 2017), we design 9 types of actions summarized in Table 3.1. An example for parsing a sentence into its compact AMR graph shown in Figure 3.3 is provided in Table 3.2.

- **SHIFT**: removes an item from the front of the buffer and pushes it to the stack.
- **REDUCE**: pops the item on top of the stack.

Action	Stack	Buffer	Concept/Relation
		Iftik, Ahmed, is, Pakistani, official, $	-
SHIFT	*Iftik*	*Ahmed, is, Pakistani, official, $*	-
MERGE		*(Iftik, Ahmed), is, Pakistani, official, $*	-
ENTITY (ENperson)		ENperson, *is, Pakistani, official, $*	ENperson
SHIFT	ENperson	*is, Pakistani, official, $*	-
SHIFT	ENperson, *is*	*Pakistani, official, $*	-
ENTITY (ENcountry)	ENperson, *is*	ENcountry, *official, $*	ENcountry
REDUCE	ENperson	ENcountry, *official, $*	-
SHIFT	ENperson, ENcountry	*official, $*	-
GEN (person)	ENperson, ENcountry	*official*, person, *$*	person
GEN (have-org-role-91)	ENperson, ENcountry	*official*, have-org-role-91, person, *$*	have-org-role-91
PRED (official)	ENperson, ENcountry	official, have-org-role-91, person, *$*	official
SHIFT	ENperson, ENcountry, *official*	have-org-role-91, person, *$*	-
LEFTLABEL (ARG2)	ENperson, ENcountry, *official*	have-org-role-91, person, *$*	have-org-role-91 $\xrightarrow{\text{ARG2}}$ official
REDUCE	ENperson, ENcountry	have-org-role-91, person, *$*	-
LEFTLABEL (ARG1)	ENperson, ENcountry	have-org-role-91, person, *$*	have-org-role-91 $\xrightarrow{\text{ARG1}}$ ENcountry
REDUCE	ENperson	have-org-role-91, person, *$*	-
SHIFT	ENperson, have-org-role-91	person, *$*	-
LEFTLABEL (ARG0-of)	ENperson, have-org-role-91	person, *$*	person $\xrightarrow{\text{ARG0-of}}$ have-org-role-91
SWAP	have-org-role-91, ENperson	person, *$*	-
LEFTLABEL (domain)	have-org-role-91, ENperson	person, *$*	person $\xrightarrow{\text{domain}}$ ENperson
REDUCE	have-org-role-91	person, *$*	-
SHIFT	have-org-role-91, person	*$*	-
PRED (ROOT)	have-org-role-91, person	ROOT	ROOT
LEFTLABEL (root)	have-org-role-91, person	ROOT	ROOT $\xrightarrow{\text{root}}$ person
REDUCE	have-org-role-91	ROOT	-
REDUCE		ROOT	-
SHIFT	ROOT		-

TABLE 3.2: Example action sequence for parsing the sentence: *Iftik Ahmed is Pakistani official.* Here *$* is the end-of-sentence symbol.

- **RIGHTLABEL**(r): creates a relation arc from the item on top of the stack to the item at the front of the buffer. Since these two items are not removed, they can be attached by another relation arc in the future. Therefore, reentrancy is allowed.
- **LEFTLABEL**(r): creates a relation in the reverse direction as RIGHTLABEL.
- **SWAP**: swaps the top two items on the stack. This action allows non-projective relations (Nivre, 2009) and helps to introduce reentrancy. Repetitive SWAP actions are disallowed to avoid infinite swapping.
- **PRED**(n): predicts the predicate and lemma concepts corresponding to the item at the front of the buffer. This action requires a look-up table M generated during the training phase. For example, *meet* is mapped to concepts such as meet, meet-01 and meet-02 in M. If the token *meet* is at the front of the buffer, then we add all its corresponding concepts based on M to the valid action sets.
- **MERGE**: removes the item at the front of the buffer and the item on top of the stack, then a composition function is applied to merge them into a new item. The item will be pushed back to the front of the buffer. This action serves to generate non-lexical concepts, whose corresponding span is larger than two. For example, if *"Iftik"* is on top of the stack and *"Ahmed"* is at the front of the buffer, a MERGE action will be applied. This action can be applied recursively if the span is larger than two.



Iftik Ahmed is Pakistani official.

FIGURE 3.3: An example sentence with its compact AMR graph.

- **ENTITY**(ℓ): pops the item at the front of the buffer, and labels it with an entity label. This action is designed for named entities in non-lexical concepts. For example, after merging *"Iftik"* and *"Ahmed"*, ENTITY(ENperson) will be used to label the item.
- **GEN**(n): creates a non-lexical concept invoked by the item at the front of the buffer, which can be any type of concept or composed representation after MERGE transition. Then the non-lexical concept is pushed back to the buffer right after the item. This action can be applied recursively. An example is that concept have-org-role-91 is invoked by *official*. If token *official* is at the front of the buffer, GEN(have-org-role-91) will be applied.

3.3 Stack LSTM

A classifier is required to decide which action to take at each time step, given the current state.

3.3.1 Stack LSTM for AMR parsing

Stack LSTM (Dyer et al., 2015) is LSTM (Hochreiter and Schmidhuber, 1997) that allows stack operations. Using stack LSTM, each state can be represented using the contents of the stack, buffer and a list with the history of actions.

Let $\mathbf{s}_t$, $\mathbf{b}_t$ and $\mathbf{a}_t$ denote the summaries of stack, buffer and the history of actions at time step t respectively. The parser state $\mathbf{y}_t$ is given by:

$$\mathbf{y}_t = \max\{\mathbf{0}, \mathbf{W}[\mathbf{s}_t; \mathbf{b}_t; \mathbf{a}_t] + \mathbf{d}\} \tag{3.1}$$

where $\mathbf{W}$ is a learned parameter matrix, and $\mathbf{d}$ is a bias term. To learn a better representation, we apply the attention mechanism (Bahdanau, Cho, and Bengio, 2015) to the words the buffer

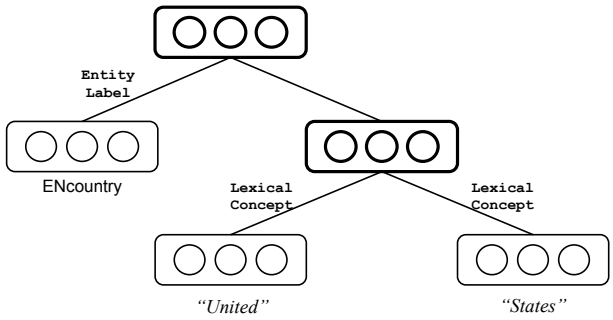


FIGURE 3.4: Example of a partial structure rooted by a non-lexical concept. *"United"* and *"States"* are merged by g_m. Then g_ℓ is applied to combine the merged representation and the label representation of ENcompany to arrive at the final representation.

contains. After getting the representation, we can use it to compute the probability p of the next action z given the parser state $\mathbf{y}_t$:

$$p(z|\mathbf{y}_t) = \frac{\exp(g_{z_t}^\top \mathbf{y}_t + q_{z_t})}{\sum_{z' \in \mathcal{A}} \exp(g_{z'}^\top \mathbf{y}_t + q_{z'})} \tag{3.2}$$

where g_z is a vector representing the embedding of the action z and q_z is a bias term. The set $\mathcal{A}$ represents the valid action set in each time step. This set varies for different parsing states due to the constraints we made for the compact AMR graph.

3.3.2 Representations and UNK strategy

Following Dyer et al. (2015), we concatenate three vector representations: pretrained word vector (Ling et al., 2015), learned word vector and learned vector for the POS tag, followed by a linear map to get the representation of each input token. For relation label representations and generated concepts, we simply use the learned embedding of the parser action that was applied to construct the relation.

Dyer et al. (2015) show that these representations can deal flexibly with out-of-vocabulary (OOV) words[5]. We extend this UNK strategy to AMR parsing. Apart from stochastically replacing (with $p = 0.2$) each singleton in the training data, we also replace the original PRED(n) as PRED(*UNK*) if the token should be selected as a concept. At the postprocessing stage, if an AMR concept is generated by PRED(*UNK*) we will use its lemma form for nouns or the most frequent sense for verbs to replace the "UNK" placeholder. This strategy makes the classifier trained not only to predict whether an OOV word should be selected as an AMR concept but also to predict the correct concept for the in-vocabulary word.

3.3.3 Composition Function

We use recursive neural networks (Socher, Bauer, Manning, et al., 2013) to compute the representations of partially constructed structures.

[5]Both OOV words in the parsing data and OOV words in the pretraining language model can be represented.

For relation attachments, a composition function g_r is used to combine representations of AMR parent concept ($\mathbf{h}$), child concept ($\mathbf{d}$) and the corresponding relation label ($\mathbf{r}$) as:

$$g_r(\mathbf{h}, \mathbf{d}, \mathbf{r}) = \tanh(\mathbf{U}_r[\mathbf{h}; \mathbf{d}; \mathbf{r}] + \mathbf{b}_r) \tag{3.3}$$

where $\mathbf{U}_r$ and $\mathbf{b}_r$ are parameters in the model, as $\mathbf{U}_m$, $\mathbf{b}_m$, $\mathbf{U}_\ell$ and $\mathbf{b}_\ell$ in equation (4) and (5).

For generated non-lexical concepts, if terminal concepts are lexical concepts as shown in Figure 3.4 we will use a composition function g_m to get their ($\mathbf{c}_1$ and $\mathbf{c}_2$) merged representation:

$$g_m(\mathbf{c}_1, \mathbf{c}_2) = \tanh(\mathbf{U}_m[\mathbf{c_1}; \mathbf{c_2}] + \mathbf{b}_m) \tag{3.4}$$

Next, another composition function g_ℓ is applied to get the representation of the labeled generated concept ($\mathbf{e}$ and its entity label $\mathbf{l}$):

$$g_\ell(\mathbf{e}, \mathbf{l}) = \tanh(\mathbf{U}_\ell[\mathbf{e}; \mathbf{l}] + \mathbf{b}_\ell) \tag{3.5}$$

3.4 Experiments

We first evaluate the effectiveness of our hybrid aligner. Then we report the final results by incorporating it into our parser. We conduct experiments on the datasets LDC2014T12, LDC2015E86 and LDC2017T10. The newswire section of LDC2014T12 is popular so we also report the performance on it. All AMR parsing results are evaluated by using Smatch (Cai and Knight, 2013).

3.4.1 Aligner Evaluation

As the hybrid aligner is designed to tackle non-projectivity caused by alignment error and null alignment, our experiments focus on these aspects. Table 3.3 shows the alignment evaluation. The ISI alignments are not included in the LDC2014T12 corpus, so we only compare JAMR and our hybrid aligner on this dataset. 2014N here refers to the newswire section of LDC2014T12.

Dataset	Aligner	F_1(%)	CR(%)	Null(%)
2014N	JAMR	92.70	14.4	5.1
	Hybrid	**96.12**	**8.8**	**2.0**
2014	JAMR	89.99	16.9	7.0
	Hybrid	**95.27**	**10.5**	**2.6**
2015	JAMR	86.23	16.5	8.9
	Hybrid	**93.05**	11.9	**2.9**
	ISI	71.92	**3.4**	20.1
2017	JAMR	84.41	13.6	11.1
	Hybrid	**92.69**	10.2	**3.4**
	ISI	75.31	**3.2**	18.2

TABLE 3.3: Alignment evaluation. F_1 indicates the final score obtained by the action sequence generated by the aligner during training process. CR(%) indicates the portion that the parser cannot handle because of non-projectivity. Null(%) indicates the percentage of AMR concepts that are not aligned.

Parser	Features				Pipeline	F_1(%)			
	POS	DEP	NER	SRL		2014N	2014	2015	2017
Flanigan et al. (2014)	√	√	×	×	Yes	59	58	-	-
artzi2015broad	√	×	×	×	Yes	67	-	-	-
Pust et al. (2015)	×	×	√	×	Yes	-	67.1	-	-
Flanigan et al. (2016a)	√	√	√	×	Yes	-	66	67	-
Zhou et al. (2016a)	√	√	√	√	No	71	66	-	-
Damonte, Cohen, and Satta (2016)	√	√	√	√	No	-	64	64	-
Wang, Xue, and Pradhan (2015b)	√	√	√	×	Yes	70	66.5	-	-
Wang and Xue (2017)	√	√	√	√	Yes	-	68.1	68.1	-
Peng et al. (2017b)	×	×	×	×	No	-	-	52	-
Konstas et al. (2017)	×	×	√	×	No	-	-	62.1	-
Ballesteros and Al-Onaizan (2017)	√	√	×	×	No	69	64	-	-
Foland and Martin (2017)	×	×	√	×	Yes	-	-	70.7±0.2	-
Buys and Blunsom (2017)	√	×	√	×	No	-	-	-	61.9
Noord and Bos (2017)	√	×	×	×	No	-	-	68.5	71.0
Peng, Gildea, and Satta (2018)	√	√	×	×	No	-	-	64	-
Vilares and Gómez-Rodríguez (2018)	√	√	√	×	No	-	-	64	-
Lyu and Titov (2018)	√	×	√	×	Yes	-	-	**73.7±0.2**	**74.4±0.2**
Groschwitz et al. (2018)	√	×	√	×	No	-	-	70.2±0.3	71.0±0.5
This work (full model)	√	×	×	×	No	**74.0±0.5**	**68.3±0.4**	68.7±0.3	69.8±0.3
no compact AMR graph	√	×	×	×	No	72.1	66.7	67.2	68.9
JAMR aligner	√	×	×	×	No	72.6	65.4	65.8	66.3
no compact AMR graph, *JAMR* aligner	√	×	×	×	No	70.3	63.9	64.6	65.3
no UNK strategy	√	×	×	×	No	72.2	66.4	67.2	68.8
no POS	×	×	×	×	No	72.4	66.9	67.3	68.8

TABLE 3.4: Parser evaluation. POS: POS tags; DEP: dependency trees; NER: named entities; SRL: semantic role labels. JAMR aligner: parser is trained by action sequence generated by the JAMR aligner. no UNK strategy: results without our UNK strategy. no compact AMR graph: results without constraints on the compact AMR graph. Standard deviation is also reported.

Non-Projectivity Issue From Table 3.3, we find that the percentage of graphs that our hybrid aligner cannot recover is significantly less on these datasets when compared to the use of the JAMR and ISI aligners. We find that the percentage drops less on LDC2017T10. The reason behind this is that many AMR concepts do not have alignments (11.1%) if we use JAMR aligner. The probability of crossing arcs drops owing to the decrease of identified concepts. The ISI aligner shares the same issue. It yields the lowest crossing arcs rate since lots of concepts are not aligned.

Empty Alignment Issue On the other hand, our hybrid aligner yields less empty alignments. This potentially helps our parser to achieve better performance, since it predicts lexical concepts solely relying on the look-up table generated by the aligner. If we do not have alignment between a certain token and its concept on the training set, the parser is less likely to predict that token as a concept during the testing phase.

Improvements on these two aspects allow the aligner to yield better action sequences for the purposes of training our parser. From Table 3.3, we can see that if we follow the action sequence generated by the hybrid aligner, our parser will achieve a higher Smatch score consistently. Improvement gain gets larger as the corpus gets larger.

3.4.2 Parser Evaluation

Then we evaluate our parser on the same datasets. As illustrated in Table 3.4, most models incorporate additional features such as dependency trees, named entities, non-lexical role labels

and external corpora. As the stack LSTM uses POS tags to get the representation of the input, our parser does not use external resources other than POS tags obtained by using NLTK (Loper and Bird, 2002). We also try to evaluate our parser by removing the POS tags. The performance drops around 1.2 F_1 points on average. POS tags help the parser to select the correct sense when PRED action is applied.

Comparisons with other parsers Currently, all state-of-the-art models either have a relatively high complexity or adopt a pipeline approach. For example, the first AMR parser JAMR (Flanigan et al., 2014) and its updated version (Flanigan et al., 2016a). The relation identification stage of JAMR has the complexity of $O(|V|^2 \log |V|)$, where $|V|$ is the number of concepts. RIGA (Barzdins and Gosko, 2016) is an ensemble system that combines CAMR and seq2seq models. Wang and Xue (2017) augment the CAMR parser (Wang, Xue, and Pradhan, 2015b) by incorporating a module called Factor Concept Labels consisting of bi-directional LSTM and CNN. Groschwitz et al. (2018) view AMR graph as the structure *AM algebra* defined in (Groschwitz et al., 2017). Since *AM algebra* can be viewed as dependency trees over the sentence, they can train a dependency parser to map the sentence into this structure. Different from the structure used in CAMR (Wang, Xue, and Pradhan, 2015b), this structure can be directly transformed to AMR graph by using postprocessing rather than relying on another transition-based system. The complexity of their projective decoder is $O(n^5)$, where n is the length of the sentence. Zhou et al. (2016a) is a graph-based parser, which adopts and improves beam search strategy. In contrast, our *linear* time system does not use a beam, but we anticipate improved results when a beam is used.

For seq2seq models, they generally require less features and build the AMR graph in an end-to-end way. However, these models usually suffer from the data sparsity issue (Peng et al., 2017b). In order to address this issue, these models utilize external corpora. Konstas et al. (2017) achieves 62.1 score by using 20M unlabeled Gigaword sentences for paired training. Noord and Bos (2017) use an additional training corpus of 100k sentences called Silver generated by an ensemble system consisting of JAMR and CAMR parsers. Without training on this additional dataset, the performance of their model is 64.0 on LDC2015E86.

Our end-to-end parser has *linear* time complexity and it exhibits competitive results on the datasets with only POS tags as additional features. Foland and Martin (2017) only use named entities as an external resource and they report the second highest F_1 on LDC2015E86. Their system also adopts a pipeline approach. The concept identification phase requires 5 different LSTMs to discover different kinds of concepts based on carefully designed features. Then they connect these components into a single graph. Unlike previous work, the very recent work by Lyu and Titov (2018) treat the alignments as latent variables in a joint probabilistic model, which improves the parsing performance substantially. They report the highest scores on LDC2015E86 and LDC2017T10 datasets. Their parser requires 5 different BiLSTMs for concept identification, alignment prediction, relation identification and root identification. Though our parser constructs the AMR graph in an end-to-end fashion, it can achieve 68.7 and 69.8 Smatch score respectively on the same test set with a simple architecture.

Fine-Grained Analysis In order to investigate how our parser performs on predicting *Named Entities*, *Reentrancies*, *Concepts*, *Negations*, etc, we also use the fine-grained evaluation tool (Damonte, Cohen, and Satta, 2016) and compare to systems which reported these scores. The results are shown in Table 3.5. We obtain relatively high results for *Concepts*, *Named Entities*

Metric	LDC2015E86				LDC2017T10			
	D'16	**V'18**	**J'18**	**Ours**	**vN'17**	**L'18**	**J'18**	**Ours**
Smatch	64	64	**70**	69	71	**74**	71	70
Unlabeled	69	68	**73**	72	74	**77**	74	73
No WSD	65	65	**71**	69	72	**76**	72	71
Reentrancies	41	44	**46**	**46**	**52**	**52**	49	49
Concepts	**83**	**83**	**83**	**83**	82	**86**	**86**	84
Named Ent.	**83**	**83**	79	80	79	**86**	78	80
Wikification	64	70	71	68	65	**76**	71	70
Negations	48	47	**52**	49	**62**	58	57	48
SRL	56	57	**63**	61	66	**70**	64	63

TABLE 3.5: Detailed results for the LDC2015E86 and LDC2017T10 test set. W'15 is Wang, Xue, and Pradhan (2015b)'s parser. F'16 is Flanigan et al. (2016a)'s parser, D'16 is Damonte, Cohen, and Satta (2016)'s parser. V'18 is Vilares and Gómez-Rodríguez (2018)'s parser. J'18 is Groschwitz et al. (2018)'s parser. vN'17 is Noord and Bos (2017)'s parser with 100K additional training pairs. L'18 is Lyu and Titov (2018)'s parser.

and *Wikification*. Also, we achieve good performance on reentrancy identification though we remove many reentrant edges during training. This indicates that the compact AMR graph encodes necessary information. On the other hand, our model does not perform so well on predicating *Negations*. We believe one reason is that our parser is a word-level model. It is hard for it to capture morphological features such as prefixes "un", "in", "il", etc. We anticipate better performance when the character-based representations are used.

Effect of Compact AMR Graph In order to better investigate the effect of different modules used in the parser, we also evaluate our parser by removing certain modules. Because our transition system is built based on the compact AMR graph, we could not evaluate the parser by isolating this representation completely. Therefore, we choose to remove some constraints defined on the compact AMR graph to investigate its effect. We can see that our representation improves the performance of our parser on three datasets, which indicates that a refined search space is beneficial to our system. When our parser is trained on the action sequence generated by the JAMR aligner as previous models, it still achieves a competitive score especially on the newswire section.

Effect of Hybrid Aligner Experiments also illustrate that the hybrid aligner consistently helps our parser. When we train our parser by using the action sequence generated by the JAMR aligner, we still achieve competitive results on the newswire section. However, the performance drops as the corpus gets larger. The largest drop occurs when we apply the parser on LDC2017T10, on which correct senses of predicates cannot be selected for many unseen words. We hypothesize that it is because the quality of alignment degrades as the corpus gets larger, which prevents the parser from learning how to find a good path in the search space during training. The hybrid aligner can alleviate this issue by generating a look-up table that has broader coverage. After incorporating the hybrid aligner, our parser achieves the best results of 68.3 on the full test set of LDC2014T12.

Chapter 4

Encoding Graph for Natural Language Generation

4.1 Introduction

In this chapter, we aim to develop efficient convolutional graph encoders for graph-to-sequence learning task. As discussed in Section 2.2.4, there exist two main challenges in building the convolutional graph encoder.

The first one is how to train a deeper GCN for modeling large and complex graphs. In order to address this issue, we introduce dense connectivity to GCNs and propose the novel densely connected graph convolutional networks (DCGCNs) as presented in Section 4.3. With the help of dense connections, we are able to train multi-layer GCN models with a large depth, allowing rich structural information to be captured for learning a better graph representation than those learned from the shallower GCN models. Experiments show that our model is able to achieve better performance for graph-to-sequence learning tasks. For the AMR-to-text generation task, our model surpasses the current state-of-the-art neural models trained on LDC2015E86 and LDC2017T10 by 2 and 4.3 BLEU points, respectively. For the syntax-based neural machine translation task, our model is also consistently better than others, showing the effectiveness of the model on a large training set.

The second challenge is how to capture non-local information in graph convolutional operations. To answer this research question, we propose to better integrate high-order information, by introducing a novel dynamic fusion mechanism and propose the Lightweight, Dynamic Graph Convolutional Networks (LDGCNs) in Section 4.5. With the help of the dynamic mechanism, LDGCNs can effectively synthesize information from different orders to model complex interactions in the AMR graph for text generation. Also, LDGCNs require no additional computational overhead, in contrast to vanilla GCN models. We further develop two novel weight sharing strategies based on the group graph convolutions and weight tied convolutions. These strategies allow the LDGCN model to reduce memory usage and model complexity. Experiments on AMR-to-text generation show that LDGCNs outperform best reported GCNs and SANs trained on LDC2015E86 and LDC2017T10 with significantly fewer parameters. On the large-scale semi-supervised setting, our model is also consistently better than others, showing the effectiveness of the model on a large training set.

4.2 Related Work

Our work builds on a rich line of recent efforts on graph-to-sequence models and graph convolutional networks.

4.2.1 Graph-to-Sequence Learning

Early efforts for graph-to-sequence learning are based on statistical methods. Lu, Ng, and Lee (2009) present a language generation model using the tree-structured meaning representation based on tree conditional random fields. Lu and Ng (2011) propose a model for language generation from lambda calculus expressions which can be represented as forest structures. Konstas and Lapata (2012) and Konstas and Lapata (2013) leverage hypergraphs for concept-to-text generation. Flanigan et al. (2016b) transform a given AMR graph into a spanning tree, before translating it into a sentence using a tree-to-string transducer. Pourdamghani, Knight, and Hermjakob (2016) adopt a phrase-based model for machine translation (Koehn, Och, and Marcu, 2003) based on a linearized AMR graph. Song et al. (2017) leverage a synchronous node replacement grammar. Konstas et al. (2017) also linearize the input graph and feed it to the Seq2Seq model (Sutskever, Vinyals, and Le, 2014; Bahdanau, Cho, and Bengio, 2015).

Sequence based neural networks may lose structural information from the original graph since they require linearization of the input graph. Recent research efforts consider developing encoders with graph neural networks. Beck, Haffari, and Cohn (2018) and Ribeiro, Gardent, and Gurevych (2019) employ gated graph neural networks (GGNNs; (Li et al., 2016b)) as the encoder and introduce the Levi graph that allows nodes and edges to have their own hidden representations. Song et al. (2018a) propose the graph recurrent networks to directly encode graph-level semantics. In order to capture non-local information, the encoder performs graph state transition by information exchange between connected nodes. Their work belongs to the family of recurrent neural networks (RNNs). Our graph encoder is built based on the graph convolutional networks (GCNs). Recurrent graph neural networks (Li et al., 2016b; Song et al., 2018a) use gated operations to update node states while graph convolutional networks use linear transformation. The contrast between our model and theirs is reminiscent of the contrast between CNNs and RNNs.

Closest to our work, Bastings et al. (2017) and Damonte and Cohen (2019) stack GCNs upon a RNN or CNN encoder since 2-layer GCNs may not be able to capture non-local information, especially when the graph is large. Marcheggiani and Perez-Beltrachini (2018) also leverages a purely GCN model for encoding the graph structure, but their model is still confined to relatively shallow architecture. More recently, self-attention models (Zhu et al., 2019; Cai and Lam, 2020b; Wang, Wan, and Jin, 2020) outperform GNN-based models as they are able to capture global dependencies. Unlike previous models, our local, yet efficient model, based solely on graph convolutions, outperforms competitive structured SANs while using a significantly smaller model.

4.2.2 Graph Convolutional Networks

Early efforts that attempt to extend neural networks to deal with arbitrary structured graphs are introduced by Gori, Monfardini, and Scarselli (2005) and Scarselli et al. (2009), where the states of nodes are updated based on the states of their neighbors. Bruna (2014) then applies the convolution operation on graph Laplacians to construct efficient architectures in the spectral domain. Subsequent efforts improve its computational efficiency with local spectral convolution techniques (Henaff, Bruna, and LeCun, 2015; Defferrard, Bresson, and Vandergheynst, 2016; Kipf and Welling, 2017).

Our approach is closely related to GCNs (Kipf and Welling, 2017), which restricts the filters to operate on a first-order neighborhood around each node. Recent improvements and extensions

of GCNs include using additional aggregation methods such as vertex attention (Veličković et al., 2018) or pooling mechanism (Hamilton, Ying, and Leskovec, 2017) to better summarize neighborhood states.

However, the best performance of GCNs is achieved with a 2-layer model while deeper models perform worse though they can potentially have access to more non-local information. Li, Han, and Wu (2018) shows that this issue is due to the over-smoothed output representations that impede distinguishing nodes from different clusters. Recent attempts that try to address this issue includes the use of layer-aggregation functions (Xu et al., 2018), which combine learned features from all layers, and the use of co-training and self-training mechanisms that encourage exploration on the entire graph (Li, Han, and Wu, 2018).

On the other hand, prior effort (Cai and Lam, 2020b) show that the locality property of GCN models precludes efficient non-local information propagation. Abu-El-Haija et al. (2019) further proved that vanilla GCNs are unable to capture feature differences among neighbors from different orders no matter how many layers are stacked. One potential approach is to incorporate information from higher order neighbors, which helps to facilitate non-local information aggregation for node classification (Abu-El-Haija et al., 2018; Abu-El-Haija et al., 2019; Morris et al., 2019). However, simple concatenation of different order representations may not be able to model complex interactions in semantics for text generation (Luan et al., 2019).

4.3 Densely Connected Graph Convolutional Networks

In this section, we will present the basic components used for constructing our Densely Connected GCN model, which is a multi-layer GCN model with a large depth, allowing rich structural information to be captured for learning a better graph representation than those learned from the shallower GCN models.

4.3.1 Graph Convolutional Networks

GCNs are neural networks that operate directly on graph structures (Kipf and Welling, 2017). Here we mathematically illustrate how multi-layer GCNs work on an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ and $\mathcal{E}$ are the set of nodes and edges, respectively. The convolution computation for node v at the l-th layer, which takes the input feature representation $\mathbf{h}^{(l-1)}$ as input and outputs the induced representation $\mathbf{h}_v^{(l)}$, can be defined as:

$$\mathbf{h}_v^{(l)} = \rho\Big(\sum_{u \in \mathcal{N}(v)} W^{(l)}\mathbf{h}_u^{(l-1)} + \mathbf{b}^{(l)}\Big) \tag{4.1}$$

where $W^{(l)}$ is the weight matrix, $\mathbf{b}^{(l)}$ is the bias vector, $\mathcal{N}(v)$ is the set of one-hop neighbors of node v, and ρ is an activation function (e.g., RELU; Nair and Hinton 2010). $\mathbf{h}_v^{(0)}$ is the initial input $\mathbf{x}_v$, where $\mathbf{x}_v \in \mathbb{R}^d$ and d is the input feature dimension.

GCNs with Residual Connections: Bastings et al. (2017) integrate residual connections (He et al., 2016) into GCNs to help information propagation. Specifically, each node is updated according to Equation 4.1 first and then the resulting representation is combined with the node's representation from the last iteration:

$$\mathbf{h}_v^{(l)} = \rho\Big(\sum_{u \in \mathcal{N}(v)} W^{(l)}\mathbf{h}_u^{(l-1)} + \mathbf{b}^{(l)}\Big) + \mathbf{h}_v^{(l-1)} \tag{4.2}$$

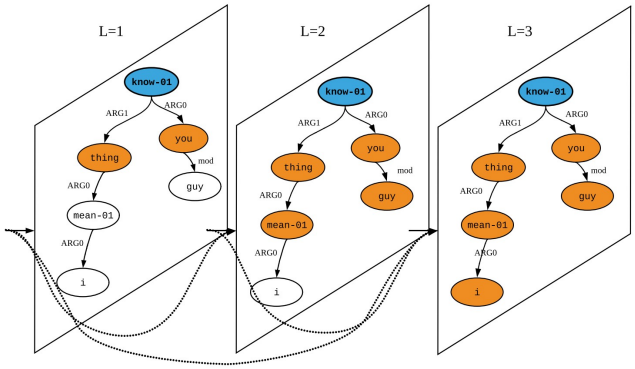


FIGURE 4.1: A 3-layer densely connected graph convolutional network. The example AMR graph here corresponds to the sentence "*You guys know what I mean.*" Each layer concatenates all preceding outputs as the input.

GCNs with Layer Aggregations: Xu et al. (2018) propose layer aggregations for GCNs, in which the final representation of each node is computed by combining the node's representations from all GCN layers:

$$\mathbf{h}_v^{final} = LA(\mathbf{h}_v^{(l)}, \mathbf{h}_v^{(l-1)}, ..., \mathbf{h}_v^{(1)}) \tag{4.3}$$

where the LA function can be concatenation, max-pooling or LSTM-attention operations (Xu et al., 2018).

4.3.2 Dense Connectivity

Dense connectivity is the core component of the proposed DCGCN. The dense connectivity strategy is illustrated in Figure 4.1 schematically. For example, the third layer receives the outputs of the first layer and the second layer, capturing the first-order, the second-order, and the third-order neighborhood information. Inspired by DenseNets (Huang et al., 2017) that distill insights from residual connections (He et al., 2016), direct connections are introduced from any layer to all its preceding layers.

With dense connectivity, node v in the l-th layer not only takes inputs from $\mathbf{h}^{(l-1)}$, but also receives information from all the preceding layers. Mathematically, we first define $\mathbf{g}_u^{(l)}$ as the concatenation of the initial node representation and the node representations produced in layers $1, \cdots, l-1$:

$$\mathbf{g}_u^{(l)} = [\mathbf{x}_u; \mathbf{h}_u^{(1)}; ...; \mathbf{h}_u^{(l-1)}] \tag{4.4}$$

Such a mechanism allows deeper layers to capture all previous information to alleviate the performance degradation in graph convolutional networks.

While dense connectivity allows training deeper neural networks, every intermediate layer is designated to be of very small size, allowing adding only a small set of features-maps at each layer. The final classifier makes predictions based on all feature-maps, which is called "collective knowledge" (Huang et al., 2017). Such a strategy improves the parameter efficiency. In practice, the dimensions of these small hidden layers d_{hidden} are decided by the number of layers L and the input feature dimension d. In DCGCN, we use $d_{hidden} = d/L$.

For example, if we have a 3-layer (L=3) DCGCN model and input dimension is 300 ($d = 300$), the hidden dimension of each layer will be $d_{hidden} = d/L = 300/3 = 100$. Then we concatenate the output of each layer to form the new representation. We have 3 layers so the output dimension is 300 (3×100). Different from the GCN model whose hidden dimension is larger than or equal to the input dimension, DCGCN model shrinks the hidden dimension as the number of layers increases in order to improve the parameter efficiency similar to DenseNets (Huang et al., 2017).

Accordingly, we modify the convolution computation of each layer as:

$$\mathbf{h}_v^{(l)} = \rho\Big(\sum_{u \in \mathcal{N}(v)} W^{(l)}\mathbf{g}_u^{(l)} + \mathbf{b}^{(l)}\Big) \tag{4.5}$$

The column dimension of the weight matrix increases by d_{hidden} per layer, i.e., $W^{(l)} \in \mathbb{R}^{d_{hidden} \times d^{(l)}}$, where $d^{(l)} = d + d_{hidden} \times (l-1)$.

Graph Attention Attention mechanisms have become almost a *de facto* standard in many sequence-based tasks (Vaswani et al., 2017). In DCGCNs, we also incorporate the self-attention strategy by implicitly specifying different weights to different nodes in a neighborhood similar to graph attention networks (Veličković et al., 2018).

In order to perform self-attention on nodes, attention coefficients are required. The input for the calculation is a set of vectors, $\mathbf{g}^{(l)} = \{\mathbf{g}_1^{(l)}, \mathbf{g}_2^{(l)}, ..., \mathbf{g}_n^{(l)}\}$, after node-wise feature transformation $\mathbf{g}_u^{(l)} = W^{(l)}\mathbf{g}_u^{(l)}$. As an initial step, a shared linear projection parameterized by a weight matrix, $W_a \in \mathbb{R}^{d_{hidden} \times d_{hidden}}$, is applied to nodes in the graph. Attention coefficients can be computed as:

$$\alpha_{ij}^{(l)} = \frac{\exp\left(\phi\left(\mathbf{a}^\top [W_a\mathbf{g}_i^{(l)}; W_a\mathbf{g}_j^{(l)}]\right)\right)}{\sum_{k \in \mathcal{N}_i} \exp\left(\phi\left(\mathbf{a}^\top [W_a\mathbf{g}_i^{(l)}; W_a\mathbf{g}_k^{(l)}]\right)\right)} \tag{4.6}$$

where $\mathbf{a} \in \mathbb{R}^{2d_{hidden}}$ is a weight vector, ϕ is the activation function (LeakyReLU; Girshick et al. 2014). These coefficients are used to compute a linear combination of the node representations. Modifying the convolution computation for attention, we arrive at:

$$\mathbf{h}_v^{(l)} = \rho\Big(\sum_{u \in \mathcal{N}(v)} \alpha_{vu}^{(l)} W^{(l)}\mathbf{g}_u^{(l)} + \mathbf{b}^{(l)}\Big) \tag{4.7}$$

where $\alpha_{vu}^{(l)}$ are normalized attention coefficients computed by the attention mechanism at the l-th layer. Note that these coefficients will not change the dimension of the output representations.

4.3.3 Convolutional Graph Encoder

We leverage DCGCNs as the graph encoder, which directly models the graph structure without linearization. Within each DCGCN block, we design two types of multi-layer DCGCNs as two sub-blocks to capture graph structure at different abstract levels. As Figure 4.2 shows, in each block, the first sub-block has n-layers and the second sub-block has m-layers. This prototype shares the same spirit with the usage of two different-sized filters in DenseNets (Huang et al., 2017). A linear transformation is employed between two sub-blocks, followed by a residual connection.

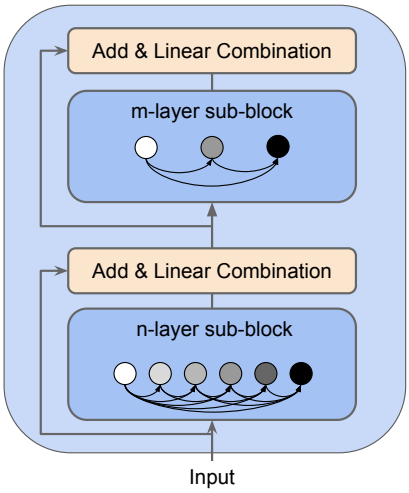


FIGURE 4.2: Each DCGCN block has two sub-blocks. Both of them are densely connected graph convolutional layers with different numbers of layers.

Linear Combination Layer In addition to densely connected layers, we include a linear combination layer between multi-layer DCGCNs to filter the representations from different DCGCNs layers, reaching a more expressive representation. This strategy is inspired by ELMo (Peters et al., 2018), which combines the hidden states from different LSTM layers. We also employ a residual connection (He et al., 2016) to incorporate the initial inputs of multi-layer GCNs into the linear combination layer, see Figure 4.5. Formally, the output of the linear combination layer is defined as:

$$\mathbf{h}_{comb} = W_{comb}\big(\mathbf{h}_{out} + \mathbf{x}_v\big) + \mathbf{b}_{comb} \tag{4.8}$$

where $\mathbf{h}_{out}$ is the output of the densely connected layers by concatenating outputs from all previous L layers $\mathbf{h}_{out} = [\mathbf{h}^{(1)}; ...; \mathbf{h}^{(L)}]$ and $\mathbf{h}_{out} \in \mathbb{R}^d$. $\mathbf{x}_v$ is the input of the DCGCN layer. $\mathbf{h}_{out}$ and $\mathbf{x}_v$ share the same dimension d. $W_{comb} \in \mathbb{R}^{d \times d}$ is a weight matrix and $\mathbf{b}_{comb}$ is a bias vector for the linear transformation. Both W_{comb} and $\mathbf{b}_{comb}$ are different according to different DCGCN layers. In addition, another linear combination layer is added to get the final representations as shown in Figure 4.5.

4.3.4 Extended Levi Graph

In order to improve the information propagation process in graph structures such as AMR graphs and dependency trees, previous researchers enrich the original input graphs with additional transformations. Marcheggiani and Titov (2017) add *reverse* edges as well as *self-loop* edges for each node to the original graph. This strategy is similar to the bidirectional recurrent neural networks (RNNs; Elman 1990) which can enjoy the information propagation from two directions. Beck, Haffari, and Cohn (2018) adapt this approach and additionally transform the directed input graphs into Levi graphs (Gross, Yellen, and Zhang, 2013). Basically, edges in the original graphs are turned into additional nodes in Levi graphs. With this approach, we can encode the original edge labels and node inputs in the same way. Specifically, Beck, Haffari, and Cohn (2018) define three types of edge labels on the Levi graph: *default*, *reverse* and *self*,

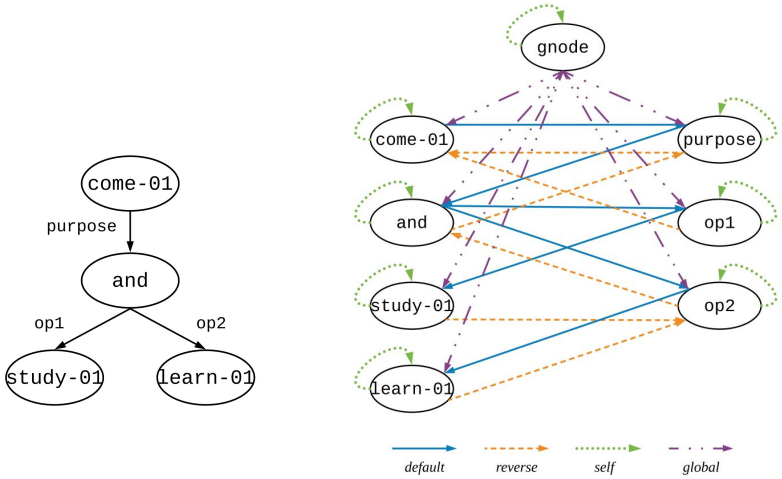


FIGURE 4.3: An AMR graph (left) and its extended Levi graph (right), which contains an additional global node and four different types of edges.

which refer to the original edges, the new virtual edges which are reverse to the original edges and the self-loop edges.

Scarselli et al. (2009) add another node that is connected to all other nodes. Zhang, Liu, and Song (2018) uses a global sentence-level node to assemble and back-distribute information. Motivated by these works, we propose **extended Levi graph**, which adds a global node in Levi graph. For every node x in the original Levi graph, there is a new edge (*global*) from the global node to x. Figure 4.3 shows an example AMR graph and its corresponding extended Levi graph. The edge type vocabulary for the extended Levi graph of the AMR graph now becomes $\mathcal{T} = \{default, reverse, self, global\}$. Our motivations are three-folds. First, the global node gives each node a global view of the input graph, which can make each node more aware of the non-local information. Second, the global node can serve as a hub to help node communications, which can facilitate the node information propagation process. Third, the output vectors of the global node in the encoder can be used as the initial states of the decoder, which are crucial for sequence-to-sequence learning tasks. Prior efforts average representations of all nodes as the graph embedding to initialize the decoder. Instead, we directly use the learned representation of the global nodes, which captures the information from all nodes in the whole graph.

The input to the syntax-based neural machine translation task is the dependency tree. Unlike the AMR graph, the sentence contains significant sequential information. Beck, Haffari, and Cohn (2018) inject this information by adding sequential connections to each token. In our model, we also add forward and backward sequential connections as illustrated in Figure 4.4. Therefore, the edge type vocabulary for the extended Levi graph of the dependency tree becomes $\mathcal{T} = \{default, reverse, self, global, forward, backward\}$.

Positional encodings about the relative or absolute position of the tokens have been proved beneficial for sequence learning (Gehring et al., 2017). We also include positional encodings by concatenating them with the learned word embeddings. The positional encodings are indexed by integer values representing the minimum distance from the root node. For example, come-01 in Figure 4.3 is the root node of the AMR graph, so its index should be 0, where and is the child node of come-01, its index is 1. Notice that we denote the index of the global node as -1.

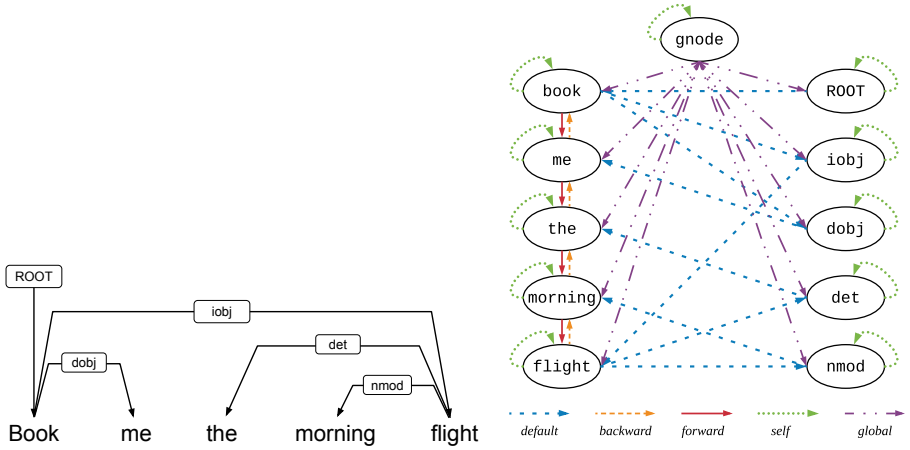


FIGURE 4.4: A dependency tree (left) and its extended Levi graph (right).

4.3.5 Direction Aggregation

Directionality and edge labels play an important role in linguistic structures. Information from incoming edges, outgoing edges and self edges should be treated differently by using separate weight matrices. Moreover, information from incoming edges that have different labels should have different weight matrices too. Following this motivation, we incorporate the directionality of an edge directly in its label. For example, node learn-01 in Figure 4.3 has three incoming edges, these edges have three different types: $default$ (from node op2), $self$ (from node learn-01) and $global$ (from node gnode). For the AMR graph, we have four types of edges while for dependency trees we have six as mentioned in Section 4.3.4. Thus, considering different types of edges, we modify the convolution computation as:

$$\mathbf{v}_t^{(l)} = \rho\Big(\sum_{\substack{u \in \mathcal{N}(v) \\ dir(u,v)=t}} \alpha_{vu}^{(l)} W_t^{(l)} \mathbf{g}_u^{(l)} + \mathbf{b}_t^{(l)} \Big) \tag{4.9}$$

where $dir(u, v)$ selects the weight matrix and bias term associated with the edge type t. For example, in the AMR generation task, there are four edge types: *default*, *reverse*, *self* and *global*. Each type corresponds to a separate weight matrix and a separate bias term.

Now we need to aggregate representations learned from different types of edges. A simple way to do this is averaging them to get the final representations. However, Hamilton, Ying, and Leskovec (2017) show that using a mean-based function to aggregate feature information from different nodes may not be satisfactory, since information from different sources should not be treated equally. Thus we assign different weights to information from different types of edges to integrate such information. Specifically, we concatenate the learned representations from all types of edges and perform a linear transformation, mathematically illustrated as:

$$f([\mathbf{v}_1^{(l)}; \cdots ; \mathbf{v}_T^{(l)}]) = W_f[\mathbf{v}_1^{(l)}; \cdots ; \mathbf{v}_T^{(l)}] + \mathbf{b}_f \tag{4.10}$$

where $W_f \in \mathbb{R}^{d' \times d_{hidden}}$ is the weight matrix and $d' = T \times d_{hidden}$. T is the size of the edge type vocabulary and d_{hidden} is the hidden dimension in DCGCN layers as described in Section

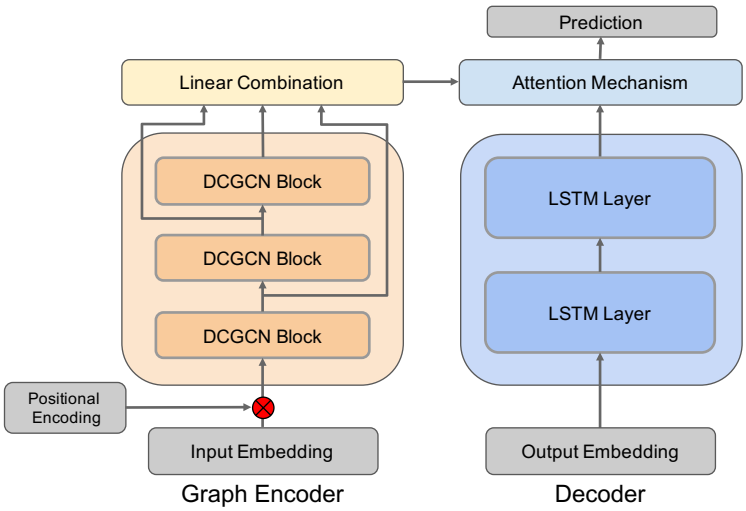


FIGURE 4.5: Architecture of the proposed graph-to-sequence model.

4.3.2. $\mathbf{b}_f \in \mathbb{R}^{d_{hidden}}$ is a bias vector. Finally, the convolution computation becomes:

$$\mathbf{h}_v^{(l)} = \rho\Big(f([\mathbf{v}_1^{(l)}; \cdots ; \mathbf{v}_T^{(l)}])\Big) \tag{4.11}$$

4.3.6 Graph-to-Sequence Model

In the following we will explain the model architecture of the graph-to-sequence model. As shown in Figure 4.5, the graph encoder is composed of DCGCN blocks and the decoder is attention-based LSTM (Bahdanau, Cho, and Bengio, 2015). The model first concatenates node embeddings and positional embeddings as inputs. The graph encoder contains a stack of N identical blocks. Then a linear transformation layer is used to combine output of all blocks into hidden representations. These representations are fed into an attention mechanism, generating the context vector. The decoder, a two-layer LSTM (Hochreiter and Schmidhuber, 1997), makes predictions based on hidden representations and the context vector.

The initial state of the decoder is the representation of the global node described in Section 4.3.5. The decoder yields the natural language sequence by calculating a sequence of hidden states sequentially. Here we also include the coverage mechanism (Tu et al., 2016). Therefore, when generating the t-th token, the decoder considers five factors: the attention memory, the word embedding of the $(t-1)$-th token, the previous hidden state of LSTM, the previous context vector and the previous coverage vector.

4.4 Experiments of DCGCNs

4.4.1 Experimental Setup

We assess the effectiveness of our models on two typical graph-to-sequence learning tasks, including AMR-to-text generation and syntax-based neural machine translation (NMT). For the AMR-to-text generation task, we use two benchmarks including the LDC2015E86 (AMR1.0) and the LDC2017T10 (AMR2.0). In these datasets, each instance contains a sentence and an

Dataset	Train	Dev	Test
AMR1.0 (LDC2015E86)	16,833	1,368	1,371
AMR2.0 (LDC2017T10)	36,521	1,368	1,371
English-Czech	181,112	2,656	2,999
English-German	226,822	2,169	2,999

TABLE 4.1: The number of sentences in four datasets.

Model	T	#P	B	C
Seq2SeqB (Beck, Haffari, and Cohn, 2018)	S	28,4M	21.7	49.1
GGNN2Seq (Beck, Haffari, and Cohn, 2018)	S	28.3M	23.3	50.4
GCNSEQ (Damonte and Cohen, 2019)	S	-	24.5	-
Seq2SeqB (Beck, Haffari, and Cohn, 2018)	E	142M	26.6	52.5
GGNN2Seq (Beck, Haffari, and Cohn, 2018)	E	141M	27.5	53.5
DCGCN (ours)	S	**19.1M**	27.9	57.3
	E	92.5M	**30.4**	**59.6**

TABLE 4.2: Main results on AMR2.0. #P shows the model size in terms of parameters; "S" and "E" denote single and ensemble models, respectively.

AMR graph. We follow Konstas et al. (2017) to apply entity simplification in the preprocessing steps. We then transform each preprocessed AMR graph into its extended Levi graph as described in Section 4.3.5. For the syntax-based NMT task, we evaluate our model on both the En-De and the En-Cs News Commentary v11 dataset from the WMT16 translation task[1]. We parse English sentences after tokenization to generate the dependency trees on the source side using SyntaxNet (Alberti et al., 2017)[2]. We tokenzie Czech and German using the Moses tokenizer[3]. On the target side, we use byte-pair encodings (BPE; Sennrich, Haddow, and Birch 2016) with 8,000 merge operations to obtain subwords. We transform the labelled dependency trees into their corresponding extended Levi graphs as described in Section 4.3.5. Table 4.1 shows the statistics of these four datasets.

We tune model hyper-parameters using random layouts based on the results of the development set. We choose the number of DCGCN blocks ($Block$) from $\{1, 2, 3, 4\}$. We select the feature dimension d from $\{180, 240, 300, 360, 420\}$. We do not use pretrained embeddings. The encoder and the decoder share the training vocabulary. We adopt Adam (Kingma and Ba, 2015) with an initial learning rate 0.0003 as the optimizer. The batch size ($Batch$) candidates are $\{16, 20, 24\}$. We determine when to stop training based on the perplexity change in the development set. For decoding, we use beam search with beam size 10. Through preliminary experiments, we find that the combinations ($Block$=4, d=360, $Batch$=16) and ($Block$=2, d=360, $Batch$=24) give best results on AMR and NMT tasks, respectively. Following previous works, we evaluate the results in terms of both BLEU (B) scores (Papineni et al., 2002) and sentence-level CHRF++ (C) scores (Popovic, 2017; Beck, Haffari, and Cohn, 2018). Particularly, we use case insensitive BLEU scores for AMR and case sensitive BLEU scores for NMT. For ensemble models, we train five models with different random seeds and then use Sockeye (Felix et al., 2017) to perform default ensemble decoding.

[1] http://www.statmt.org/wmt16/translation-task.html
[2] https://github.com/tensorflow/models/tree/master/research/syntaxnet
[3] https://github.com/moses-smt/mosesdecoder

Model	**External**	B
Seq2SeqK (Konstas et al., 2017)	-	22.0
GraphLSTM (Song et al., 2018a)	-	23.3
GCNSEQ (Damonte and Cohen, 2019)	-	24.4
DCGCN(single)	-	25.9
DCGCN(ensemble)	-	**28.2**
TSP (Song et al., 2016)	ALL	22.4
PBMT (Pourdamghani, Knight, and Hermjakob, 2016)	ALL	26.9
Tree2Str (Flanigan et al., 2016b)	ALL	23.0
SNRG (Song et al., 2017)	ALL	25.6
Seq2SeqK (Konstas et al., 2017)	0.2M	27.4
GraphLSTM (Song et al., 2018a)	0.2M	28.2
DCGCN(single)	0.1M	29.0
DCGCN(single)	0.2M	**31.6**
Seq2SeqK (Konstas et al., 2017)	2M	32.3
GraphLSTM (Song et al., 2018a)	2M	33.6
Seq2SeqK (Konstas et al., 2017)	20M	33.8
DCGCN(single)	0.3M	33.2
DCGCN(ensemble)	0.3M	**35.3**

TABLE 4.3: Main results on AMR1.0 with/without external data.

4.4.2 Main Results on AMR-to-text Generation

We compare the performance of DCGCNs with the other three kinds of models: (1) sequence-to-sequence (Seq2Seq) models which use linearized graphs as inputs; (2) graph encoders (GGNN2Seq, GraphLSTM, GCNSEQ); (3) models trained with external resources. For convenience, we denote the LSTM-based Seq2Seq models of Konstas et al. (2017) and Beck, Haffari, and Cohn (2018) as Seq2SeqK and Seq2SeqB, respectively. GGNN2Seq (Beck, Haffari, and Cohn, 2018) is the model that leverages GGNNs as graph encoders. GCNSEQ (Damonte and Cohen, 2019) applies a bidirectional RNN on top of the 2-layer GCNs as the encoder.

Table 4.2 shows the results on AMR2.0. Our single model achieves 27.6 BLEU points, which is the new state-of-the-art result for single models. In particular, our single DCGCN model consistently outperforms Seq2Seq models by a significant margin when trained without external resources. For example, the single DCGCN model gains 5.9 more BLEU points than the single models of Seq2SeqB on AMR2.0. These results demonstrate the importance of explicitly capturing the graph structure in the encoder.

In addition, our single DCGCN model obtains better results than previous ensemble models. For example, on AMR2.0, the single DCGCN model is 1 BLEU point higher than the ensemble model of Seq2SeqB. Our model requires substantially fewer parameters, e.g., the parameter size is only 3/5 and 1/9 of those in GGNN2Seq and Seq2SeqB, respectively. The ensemble approach based on combining five DCGCN models initialized with different random seeds achieves a BLEU score of 30.4 and a CHRF++ score of 59.6.

Under the same setting, our model also consistently outperforms graph encoders based on recurrent neural networks or gating mechanisms. For GGNN2Seq, our single model is 3.3 and 0.1 BLEU points higher than their single and ensemble models, respectively. We also have similar observations in term of CHRF++ scores for sentence-level evaluations. DCGCN also

Model	T	English-German			English-Czech		
		#P	B	C	#P	B	C
BoW+GCN (Bastings et al., 2017)	S	-	12.2	-	-	7.5	-
CNN+GCN (Bastings et al., 2017)	S	-	13.7	-	-	8.7	-
BiRNN+GCN (Bastings et al., 2017)	S	-	16.1	-	-	9.6	-
PB-SMT (Beck, Haffari, and Cohn, 2018)	S	-	12.8	43.2	-	8.6	36.4
Seq2SeqB (Beck, Haffari, and Cohn, 2018)	S	41.4M	15.5	40.8	39.1M	8.9	33.8
GGNN2Seq (Beck, Haffari, and Cohn, 2018)	S	41.2M	16.7	42.4	38.8M	9.8	33.3
DCGCN (ours)	S	**29.7M**	**19.0**	**44.1**	**28.3M**	**12.1**	**37.1**
Seq2SeqB (Beck, Haffari, and Cohn, 2018)	E	207M	19.0	44.1	195M	11.3	36.4
GGNN2Seq (Beck, Haffari, and Cohn, 2018)	E	206M	19.6	45.1	194M	11.7	35.9
DCGCN (ours)	E	**149M**	**20.5**	**45.8**	**142M**	**13.1**	**37.8**

TABLE 4.4: Main results on English-German and English-Czech datasets.

outperforms GraphLSTM by 2.0 BLEU points in the fully supervised setting as shown in Table 4.3. Note that GraphLSTM uses char-level neural representations and pretrained word embeddings, while our model solely relies on word-level representations with random initializations. This empirically shows that compared to recurrent graph encoders, DCGCNs can learn better representations for graphs. For GCNSEQ, our single models are 3.1 and 1.3 BLEU points higher than their models trained on AMR2.0 and AMR1.0 dataset, respectively. These results demonstrate that DCGCNs can capture contextual information without relying on additional RNNs.

Moreover, we compare our results with the state-of-the-art semi-supervised models on the AMR1.0 (Table 4.3), including non-neural methods such as TSP (Song et al., 2016), PBMT (Pourdamghani, Knight, and Hermjakob, 2016), Tree2Str (Flanigan et al., 2016b) and SNRG (Song et al., 2017). All these models train language models on the whole Gigaword corpus. Our ensemble model gives 28.2 BLEU points without external data, which is better than them.

Following Konstas et al. (2017) and Song et al. (2018a), we also evaluate our model using external Gigaword sentences as training data. We first use the additional data to pretrain the model, then fine-tune it on the gold data. Using additional 0.1M data, the single DCGCN model achieves a BLEU score of 29.0, which is higher than Seq2SeqK (Konstas et al., 2017) and GraphLSTM (Song et al., 2018a) trained with 0.2M additional data. When using the same amount of 0.2M data, the performance of DCGCN is 4.2 and 3.4 BLEU points higher than Seq2SeqK and GraphLSTM. DCGCN model is able to achieve a competitive BLEU points (33.2) by using 0.3M external data, while GraphLSTM achieves a score of 33.6 by using 2M data and Seq2SeqK achieves a score of 33.8 by using 20M data. These results show that our model is more effective in terms of using automatically generated AMR graphs. Using 0.3M additional data, our ensemble model achieves the new state-of-the-art result of 35.3 BLEU points.

4.4.3 Main Results on Syntax-based NMT

Table 4.4 shows the results for the English-German (En-De) and English-Czech (En-Cs) translation tasks. BoW+GCN, CNN+GCN and BiRNN+GCN refer to employing the following encoders with a GCN layer on top respectively: 1) a bag-of-words encoder, 2) a one-layer CNN, 3) a bi-directional RNN. PB-SMT is the phrase-based statistical machine translation model using Moses (Koehn et al., 2007). Our single model achieves 19.0 and 12.1 BLEU points on the En-De and En-Cs tasks, respectively, significantly outperforming all the single models. For

Block	n	m	B	C
	1	1	17.6	48.3
	1	2	19.2	50.3
	2	1	18.4	49.1
	1	3	19.6	49.4
1	3	1	20.0	50.5
	3	3	21.4	51.0
	3	6	21.8	51.7
	6	3	21.7	51.5
	6	6	22.0	52.1
	3	6	**23.5**	53.3
2	6	3	23.3	**53.4**
	6	6	22.0	52.1

TABLE 4.5: Effect of the number of layers inside DCGCN sub-blocks.

example, compared to the best GCN-based model (BiRNN+GCN), our single DCGCN model surpasses it by 2.7 and 2.5 BLEU points on the En-De and En-Cs tasks, respectively. Our models consist of full GCN layers, removing the burden of employing a recurrent encoder to extract non-local contextual information in the bottom layers. Compared to non-GCN models, our single DCGCN model is 2.2 and 1.9 BLEU points higher than the current state-of-the-art single model (GGNN2Seq) on the En-De and En-Cs tasks, respectively. In addition, our single model is comparable to the ensemble results of Seq2SeqB and GGNN2Seq, while the number of parameters of our models is only about 1/6 of theirs. Additionally, the ensemble DCGCN models achieve 20.5 and 13.1 BLEU points on the En-De and En-Cs tasks, respectively. Our ensemble results are significantly higher than those of the state-of-the-art syntax-based ensemble models reported by GGNN2Seq (En-De: 20.5 v.s. 19.6; En-Cs: 13.1 v.s. 11.7 in terms of BLEU).

4.4.4 Additional Experiments

Layers in the sub-block. Table 4.5 shows the effect of the number of layers of each sub-block on the AMR1.0 development set. DenseNets (Huang et al., 2017) use two kinds of convolution filters: 1×1 and 3×3. Similar to DenseNets, we choose the values of n and m for layers from $[1, 2, 3, 6]$. We choose this value range by considering the scale of non-local nodes, the abstract information at different levels and the calculation efficiency. For brevity, we only show representative configurations. We first investigate DCGCN with one block. In general, the performance increases when we gradually enlarge n and m. For example, when n=1 and m=1, the BLEU score is 17.6; when n=6 and m=6, the BLEU score becomes 22.0. We observe that (n=6, m=3), (n=3, m=6) and (n=6, m=6) give similar results for both 1 and 2 DCGCN blocks. Since the first two settings contain less parameters than the third setting, it is reasonable to choose either (n=6, m=3) or (n=3, m=6). For later experiments, we use (n=6, m=3).

Comparisons with Baselines. The first block in Table 4.6 shows the performance of two baseline models: multi-layer GCNs with residual connections (GCN+RC) and multi-layer GCNs with both residual connections and layer aggregations (GCN+RC+LA). In general, increasing the number of layers from 2 to 9 boosts the model performance. However, when the layer

GCN	B	C	**GCN**	B	C
+RC (2)	16.8	48.1	+RC+LA (2)	18.3	47.9
+RC (4)	18.4	49.6	+RC+LA (4)	18.0	51.1
+RC (6)	19.9	49.7	+RC+LA (6)	21.3	50.8
+RC (9)	**21.1**	50.5	+RC+LA (9)	**22.0**	52.6
+RC (10)	20.7	**50.7**	+RC+LA (10)	21.2	**52.9**
DCGCN1 (9)	22.9	53.0	DCGCN3 (27)	24.8	54.7
DCGCN2 (18)	24.2	54.4	DCGCN4 (36)	**25.5**	**55.4**

TABLE 4.6: Comparisons with baselines. +RC denotes residual connections. +LA denotes layer aggregations. DCGCNi represents model with i blocks, containing $i \times (n + m)$ layers. The number of layers is shown in parenthesis.

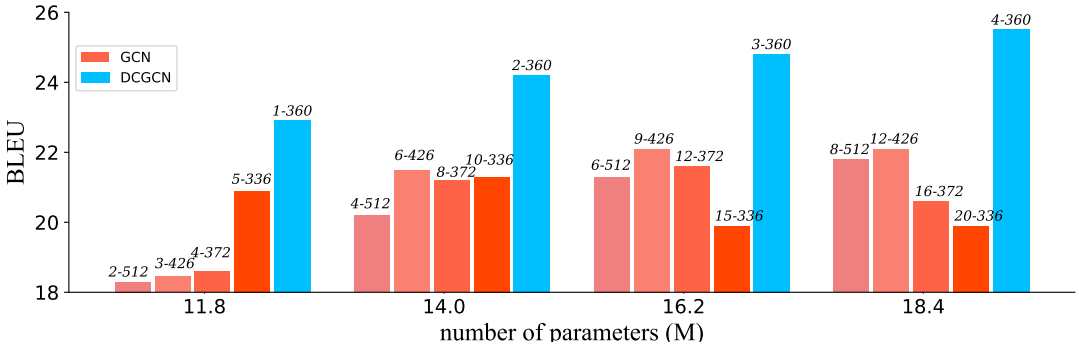


FIGURE 4.6: Comparison of DCGCN and GCN over different numbers of parameters. a-b means the model has a layers (a blocks for DCGCN) and the hidden size is b (e.g., 5-336 means a 5-layers GCN with the hidden size 336).

number exceeds 10, the performance of both baseline models start to drop. For example, GCN+RC+LA (10) achieves a BLEU score of 21.2, which is worse than GCN+RC+LA (9). In preliminary experiments, we cannot manage to train very deep GCN+RC and GCN+RC+LA models. In contrast, DCGCN models can be trained using a large number of layers. For example, DCGCN4 contains 36 layers. When we increase the DCGCN blocks from 1 to 4, the model performance continues increasing on the AMR1.0 development set. We therefore choose DCGCN4 for AMR experiments. Using a similar method, DCGCN2 is selected for NMT tasks. When the layer numbers are 9, DCGCN1 is better than GCN+RC in terms of B/C scores (21.7/51.5 v.s. 21.1/50.5). GCN+RC+LA (9) is slightly better than DCGCN1. However, when we set the number to 18, GCN+RC+LA achieves a BLEU score of 19.4, which is significantly worse than the BLEU score obtained by DCGCN2 (23.3). We also try GCN+RC+LA (27), but it does not converge. In conclusion, these results show the robustness and effectiveness of DCGCN models.

Performance v.s. Parameter Budget. We also evaluate the performance of DCGCN model against different numbers of parameters on the AMR generation task. Results are shown in Figure 4.6. Specifically, we try four parameter budgets, including 11.8M, 14.0M, 16.2M and 18.4M. These numbers correspond to the model size (in terms of number of parameters) of DCGCN1, DCGCN2, DCGCN3 and DCGCN4, respectively. For each budget, we vary both the depth of GCN models and the hidden vector dimensions of each node in GCNs in order to exhaust the entire budget. For example, $GCN(2) - 512$, $GCN(3) - 426$, $GCN(4) - 372$ and

Model	D	#P	B	C
DCGCN(1)	300	10.9M	20.9	52.0
DCGCN(2)	180		**22.2**	**52.3**
DCGCN(2)	240	11.3M	22.8	52.8
DCGCN(4)	180	11.4M	**23.4**	**53.4**
DCGCN(1)	420	12.6M	22.2	52.4
DCGCN(2)	300	12.5M	23.8	53.8
DCGCN(3)	240	12.3M	**23.9**	**54.1**
DCGCN(2)	360	14.0M	24.2	**54.4**
DCGCN(3)	300		**24.4**	54.2
DCGCN(2)	420	15.6M	24.1	53.7
DCGCN(4)	300		**24.6**	**54.8**
DCGCN(3)	420	18.6M	24.5	54.6
DCGCN(4)	360	18.4M	**25.5**	**55.4**

TABLE 4.7: Comparisons of different DCGCN models under almost the same parameter budget.

Model	B	C
DCGCN4	25.5	55.4
-{4} dense block	24.8	54.9
-{3, 4} dense blocks	23.8	54.1
-{2, 3, 4} dense blocks	23.2	53.1

TABLE 4.8: Ablation study for density of connections on AMR1.0. -{i} dense block denotes removing the dense connections in the i-th block.

$GCN(5) - 336$ contain about 11.8M parameters, where $GCN(i) - d$ indicates a GCN model with i layers and the hidden size for each node is d. We compare DCGCN1 with these four models. DCGCN1 gives 22.9 BLEU points. For the GCN models, the best result is obtained by $GCN(5) - 336$, which falls behind DCGCN1 by 2.0 BLEU points. We compare DCGCN2, DCGCN3 and DCGCN4 with their equal-sized GCN models in a similar way. The results show that DCGCN consistently outperforms GCN under the same parameter budget. When the parameter budget becomes larger, we can observe that the performance difference becomes more prominent. In particular, the BLEU margins between DCGCN models and their best GCN models are 2.0, 2.7, 2.7 and 3.4, respectively.

Performance v.s. Layers. We compare DCGCN models with different layers under the same parameter budget. Table 4.7 shows the results. For example, when both DCGCN1 and DCGCN2 are limited to 10.9M parameters, DCGCN2 obtains 22.2 BLEU points, which is higher than DCGCN1 (20.9). Similarly, when DCGCN3 and DCGCN4 contain 18.6M and 18.4M parameters. DCGCN4 outperforms DCGCN3 by 1 BLEU point with a slightly smaller model. In general, we found when the parameter budget is the same, deeper DCGCN models can obtain better results than the shallower ones.

Level of Density. Table 4.8 shows the ablation study of the level of density of our model. We use DCGCNs with 4 dense blocks as the full model. Then we remove dense connections

Model	B	C
DCGCN4	25.5	55.4
Encoder Modules		
-Linear Combination	23.7	53.2
-Global Node	24.2	54.6
-Direction Aggregation	24.6	54.6
-Graph Attention	24.9	54.7
-Global Node&Linear Combination	22.9	52.4
Decoder Modules		
-Coverage Mechanism	23.8	53.0

TABLE 4.9: Ablation study for modules used in the graph-to-sequence model

gradually from the last block to the first block. In general, the performance of the model drops substantially as we remove more dense connections until it cannot converge without dense connections. The full model gives 25.5 BLEU points on the AMR1.0 dev set. After removing the dense connections in the last block, the BLEU score becomes 24.8. Without using the dense connections in the last two blocks, the score drops to 23.8. Furthermore, excluding the dense connections in the last three blocks only gives 23.2 BLEU points. Although these four models have the same number of layers, dense connections allow the model to achieve much better performance. If all the dense connections are not considered, the model does not converge at all. These results indicate dense connections do play a significant role in our model.

Ablation Study for Encoder and Decoder. Following Song et al. (2018a), we conduct a further ablation study for modules used in the graph encoder and LSTM decoder on the AMR1.0 dev set, including linear combination, global node, direction aggregation, graph attention mechanism and coverage mechanism using the 4-block models by always keeping the dense connections. Table 4.9 shows the results. For the encoder, we find that the linear combination and the global node have more contributions in terms of B/C scores. The results drop by 2/2.2 and 1.3/1.2 points respectively after removing them. Without these two components, our model gives a BLEU score of 22.6, which is still better than the best GCN+RC model (21.1) and the best GCN+RC+LA model (22.1). Adding either the global node or the linear combination improves the baseline models with only dense connections. This suggests that enriching input graphs with the global node and including the linear combination can facilitate GCNs to learn better information aggregations, producing more expressive graph representations. Results also show the linear combination is more effective than the global node. Considering them together further enhances the model performance. After removing the graph attention module, our model gives 24.9 BLEU points. Similarly, excluding the direction aggregation module leads to a performance drop to 24.6 BLEU points. The coverage mechanism is also effective in our models. Without the coverage mechanism, the result drops by 1.7/2.4 points for B/C scores.

4.4.5 Analysis and Discussion

Graph size. Following Bastings et al. (2017), we show in Figure 4.7 the CHRF++ score variations according to the graph size $|G|$ on the AMR2015 development set, where $|G|$ refers to the number of nodes in the extended Levi graph. We bin the graph size into five classes

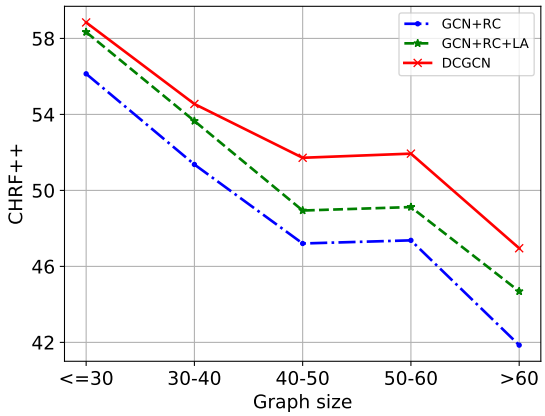


FIGURE 4.7: CHRF++ scores with respect to the input graph size.

$(\leq 30, (30, 40], (40, 50], (50, 60], > 60)$. We average the sentence-level CHRF++ scores of the sentences in the same bin to plot Figure 4.7. For small graphs (i.e., $|G| \leq 30$), DCGCN obtains similar results as the baselines. For large graphs, DCGCN significantly outperforms the two baselines. In general, as the graph size increases, the gap between DCGCN and the two baselines becomes larger. In addition, we can also notice that the margin between GCN and GCN+LA is quite stable, while the margin between DCGCN and GCN+LA varies according to the graph size. The trend for BLEU scores is similar to CHRF++ scores. This suggests that DCGCN can perform better for larger graphs as its deeper architecture can capture the long-distance dependencies. Dense connections facilitate information propagation in large graphs, while shallow GCNs might struggle to capture such dependencies.

Example output. Table 4.10 shows example outputs from three models for the AMR-to-text task, together with the corresponding AMR graph as well as the text reference. The word "technology" in the reference acts as a link between "global trade" and "weapons of mass destruction", offering the background knowledge to help understand the context. The word "instructions" also plays a crucial role in the generated sentence – without the word the sentence will have a significantly different meaning. Both GCN+RC and GCN+RC+LA fail to successfully generate these two important words. The output from GCN+RC does not even appear to be grammatically correct. In contrast, DCGCN manages to generate both words. We believe this is because DCGCN is able to learn richer semantic information by capturing complex long dependencies. GCN+RC+LA does generate an output which looks similar to the reference at the token level. However, the conveyed semantic information in the generated sentence largely differs from that of the reference. DCGCNs do not have this problem.

(s / state-01 :ARG0 (p / person :ARG0-of (h / have-org-role-91 :ARG1 (i / intelligence :mod (c / country :wiki "united_states" :name (n / name :op1 "u.s."))) :ARG2 (o / official))) :ARG1 (c2 / continue-01 :ARG0 (p2 / person :ARG0-of (h2 / have-org-role-91 :ARG2 (o2 / official :mod (c3 / country :wiki "north_korea" :name (n2 / name :op1 "north" :op2 "korea"))))) :ARG1 (t / trade-01 :ARG1 (t2 / **technology** :purpose (w / weapon :ARG2-of (d / destroy-01 :degree (m / mass)))) :mod (g / globe)) :ARG2-of (i2 / include-01 :ARG1 (i3 / **instruct-01** :ARG3 (m2 / make-01 :ARG1 (m3 / missile :ARG1-of (a / advanced-02)))))))
Reference: u.s. intelligence officials stated that north korean officials are continuing global trade in **technology** for weapons of mass destruction including **instructions** for making advanced missiles.
GCN+RC: a u.s. intelligence official stated that north korea officials continued the global trade for weapons of mass destruction by making advanced missiles to make advanced missiles.
GCN+RC+LA: a u.s. intelligence official stated that north korea officials continued global trade with weapons of mass destruction including making advanced missiles.
DCGCN: a u.s. intelligence official stated that north korea officials continue global trade on **technology** for weapons of mass destruction including **instructions** to make advanced missiles.

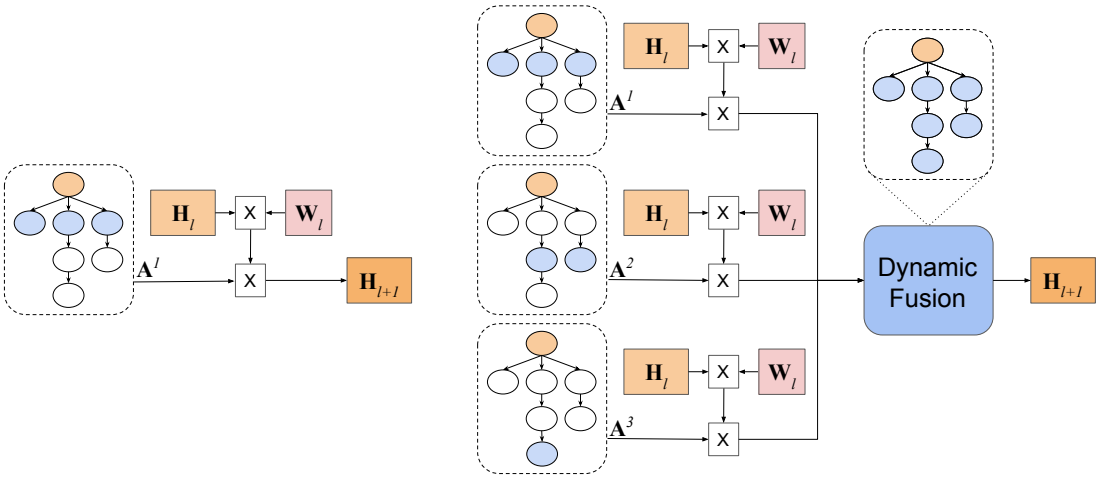
TABLE 4.10: Example outputs.

4.5 Lightweight Dynamic Graph Convolutional Networks

In this section, we will present the basic components used for constructing our lightweight dynamic GCN model. GCNs are generally more computationally efficient than structured SANs as their computation cost scales linearly and no additional relation encoders are required. However, the locality nature of GCNs precludes efficient non-local information propagation. To address this issue, we propose the dynamic fusion mechanism, which integrates higher order information for better non-local information aggregation. With the help of this mechanism, our model solely based on graph convolutions is able to outperform competitive structured SANs.

4.5.1 Dynamic Fusion Mechanism

Inspired by Gated Linear Units (GLUs; Dauphin et al. 2016), which leverage gating mechanisms (Hochreiter and Schmidhuber, 1997) to dynamically control information flows in the convolutional neural networks, we propose dynamic fusion mechanism (DFM) to integrate information from different orders. DFM allows the model to automatically synthesize information from neighbors at varying degrees of hops away. Similar to GLUs, DFM retains non-linear capabilities of the layer while allowing the gradient to propagate through the linear unit without scaling. Based on this non-linear mixture procedure, DFM is able to control the information flows from a range of orders to specific nodes in the AMR graph.



(a) Vanilla GCN Layer (b) LDGCN Layer

FIGURE 4.8: Comparison between vanilla GCNs and LDGCNs. $\mathbf{H}_l$ denotes the representation of l-th layer. $\mathbf{W}_l$ denotes the trainable weights and $\times$ denotes matrix multiplication.

The dynamic fusion mechanism is illustrated in Figure 4.8 schematically. Vanilla GCNs take the 1st-order adjacency matrix $\mathbf{A}^1$ as the input, which only captures information from one-hop neighbors. LDGCNs take k number of k-order adjacency matrix $\mathbf{A}^k$ as inputs, $\mathbf{W}_l$ is shared for all $\mathbf{A}^k$. k is set to 3 here for simplification. A dynamic fusion mechanism is applied to integrate the information from 1- to k-hop neighbors.

Formally, graph convolutions based on DFM are defined as:

$$\begin{aligned}\mathbf{H}_{l+1} =&(1-\frac{1}{K-1}\sum_{1<k<K}\mathbf{G}_l^{(k)})\odot\phi(\mathbf{A}\mathbf{H}_l\mathbf{W}_l+\mathbf{b}_l)\\&+\frac{1}{K-1}\sum_{1<k<K}\mathbf{G}_l^{(k)}\odot\phi(\mathbf{A}^k\mathbf{H}_l\mathbf{W}_l+\mathbf{b}_l)\end{aligned}\tag{4.12}$$

where $\mathbf{G}_l^{(k)}$ is a gating matrix conditioned on the k-th order adjacency matrix $\mathbf{A}^k$, namely:

$$\mathbf{G}_l^{(k)}=(1-\lambda^k)\odot\sigma(\mathbf{A}^k\mathbf{H}_l\mathbf{W}_l+\mathbf{b}_l)\tag{4.13}$$

where $\odot$ denotes elementwise product, σ denotes the sigmoid function, $\lambda\in(0,1)$ is a scalar, $K\geq 2$ is the highest order used for information aggregation, and $\mathbf{W}_l$ denotes trainable weights shared by different $\mathbf{A}^k$. Both λ and K are hyperparameters.

Computational Overhead In practice, there is no need to calculate or store $\mathbf{A}^k$. $\mathbf{A}^k\mathbf{H}_l$ is computed with right-to-left multiplication. Specifically, if k=3, we can calculate $\mathbf{A}^3\mathbf{H}_l$ using the form $(\mathbf{A}(\mathbf{A}(\mathbf{A}\mathbf{H}_l)))$. Since we store $\mathbf{A}$ as a sparse matrix with m non-zero entries as vanilla GCNs, an efficient implementation of our layer takes $O(k_{max}\times m\times d)$ computational time, where k_{max} is the highest order used and d is the feature dimension of $\mathbf{H}_l$. Under the realistic assumptions of $k_{max}\ll m$ and $d\ll m$, running an l-layer model takes $O(lm)$ computational time. This matches the computational complexity of the vanilla GCNs. On the other hand, DFM does not require additional parameters as the weight matrix is shared over various orders.

Deeper LDGCNs To further facilitate the non-local information aggregation, we stack several LDGCN layers. In order to stabilize the training, we introduce dense connections (Huang et al., 2017; Guo et al., 2019) into the LDGCN model. Mathematically, we define the input of the l-th layer $\hat{\mathbf{H}}_l$ as the concatenation of all node representations produced in layers $1,\cdots,l-1$:

$$\hat{\mathbf{H}}_l=[\mathbf{H}_0;\mathbf{H}_1;...;\mathbf{H}_{l-1}]\tag{4.14}$$

Accordingly, $\mathbf{H}_l$ in Equation 4.12 is replaced by $\hat{\mathbf{H}}_l$.

$$\begin{aligned}\mathbf{H}_{l+1} =&(1-\frac{1}{K-1}\sum_{1<k<K}\mathbf{G}_l^{(k)})\odot\phi(\mathbf{A}\hat{\mathbf{H}}_l\mathbf{W}_l+\mathbf{b}_l)\\&+\frac{1}{K-1}\sum_{1<k<K}\mathbf{G}_l^{(k)}\odot\phi(\mathbf{A}^k\hat{\mathbf{H}}_l\mathbf{W}_l+\mathbf{b}_l)\end{aligned}\tag{4.15}$$

where $\mathbf{W}_l\in\mathbb{R}^{d_l\times d}$ and d_l=$d\times(l-1)$. The model size scales linearly as we increase the depth of the network.

Although we are able to train a very deep LDGCN model, the LDGCN model size increases sharply as we stack more layers, resulting in large model complexity. To maintain a better balance between parameter efficiency and model capacity, we develop two novel parameter saving strategies. We first reduce partial parameters in each layer based on group graph convolutions. Then we further share parameters across all layers based on weight tied convolutions. These strategies allow the LDGCN model to reduce memory usage and model complexity.

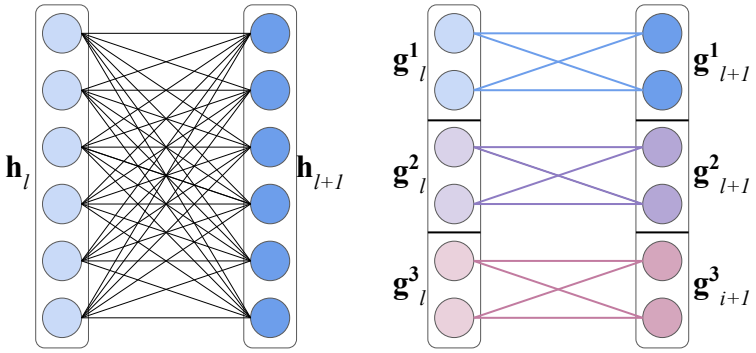


FIGURE 4.9: Comparison between vanilla graph convolutions and depthwise graph convolutions. The input and output representation of the l-th layer $\mathbf{h}_l$ and $\mathbf{h}_{l+1}$ are partitioned into N=3 disjoint groups.

4.5.2 Group Graph Convolutions

Group convolutions have been used to build efficient networks for various computer vision tasks as they can better integrate feature maps (Xie et al., 2017; Li et al., 2019) and have lower computational costs (Howard et al., 2017; Zhang et al., 2017a) compared to vanilla convolutions. In order to reduce the model complexity in the deep LDGCN model, we extend group convolutions to GCNs by introducing group convolutions along two directions: depthwise and layerwise.

Depthwise Graph Convolutions: Graph convolutions operate on the features of n nodes $\mathbf{H} \in \mathbb{R}^{n \times d}$. For simplicity, we assume n=1, the input and output representation of the l-th layer are $\mathbf{h}_l \in \mathbb{R}^{d_l}$ and $\mathbf{h}_{l+1} \in \mathbb{R}^{d_{l+1}}$, respectively. As shown in Figure 4.9, the size of the weight matrix $\mathbf{W}_l$ in a vanilla graph convolutions is $d_l \times d_{l+1}$. In depthwise graph convolutions, $\mathbf{h}_l$ is partitioned into N mutually exclusive groups $\{\mathbf{g}^1_l,...,\mathbf{g}^N_l\}$. The weight $\mathbf{W}_l$ of each layer is also partitioned into N mutually exclusive groups $\mathbf{W}^1_l,...,\mathbf{W}^N_l$. The dimension of each weight is $\frac{d_l}{N} \times \frac{d_{l+1}}{N}$. Finally, we obtain the output representation $\mathbf{h}_{l+1}$ by concatenating N groups of outputs $[\mathbf{g}^1_{l+1};...;\mathbf{g}^N_{l+1}]$. Now the parameters of each layer can be reduced by a factor of N, to $\frac{d_l \times d_{l+1}}{N}$.

Layerwise Graph Convolutions: These group convolutions are built based on densely connected graph convolutions (Guo et al., 2019). As shown in Figure 4.10, each layer takes the concatenation of outputs from all preceding layers as its input. For example, layer L_2 takes the concatenation of $[\mathbf{h}_0; \mathbf{h}_1]$ as its input. Guo et al. (2019) further adopt a dimension shrinkage strategy. Assume $\mathbf{h}_0 \in \mathbb{R}^d$ and that the network has L layers. The dimension of output for each layer is set to $\frac{d}{L}$. Finally, we concatenate the output of L layers $[\mathbf{h}_1; ...; \mathbf{h}_L]$ to form the final representation $h_{final} \in \mathbb{R}^d$. Therefore, the size of the weight matrix for the l-th layer is $(d + \frac{d \times (l-1)}{L}) \times \frac{d}{L}$.

Notice that main computation cost originates in the computation of $\mathbf{h}_0$ as it has a large dimension and it is concatenated to the input of each layer. In layerwise graph convolutions however, we improve the parameter efficiency by dividing the input representation $\mathbf{h}_0$ into M

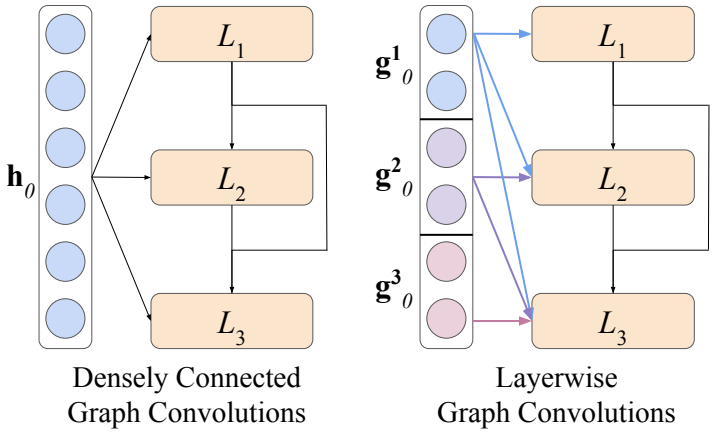


FIGURE 4.10: Comparison between vanilla and layerwise graph convolutions. The input representation $\mathbf{h}_0$ is partitioned into M=3 disjoint groups.

groups $\{\mathbf{g}_0^1,...\mathbf{g}_0^M\}$, where M equals the total number of layers L. The first group $\mathbf{g}_0^1$ is fed to all L layers, and the second group $\mathbf{g}_0^2$ is fed to (L-1) layers, so on and so forth. Accordingly, the size of weight matrix for the l-th layer is $(\frac{d\times(2l-1)}{L}) \times \frac{d}{L}$.

Formally, we partition the input representations of n concept $\mathbf{H}_0 \in \mathbb{R}^{n\times d}$ to the first layer into M groups $\{\mathbf{G}_0^1, ..., \mathbf{G}_0^M\}$, where the size of each group is $n \times \frac{d}{M}$. Accordingly, we modify the input of the l-th layer $\hat{\mathbf{H}}_l$ in Equation 4.14 as:

$$\hat{\mathbf{H}}_l = [\mathbf{G}_0^1; ...; \mathbf{G}_0^M; \mathbf{H}_1; ...; \mathbf{H}_{l-1}] \tag{4.16}$$

In practice, we combine these two convolutions together to further reduce the model size. For example, assume the size of the input is d=360 and the number of layers is L=6. The size of the weight matrix for the first layer (l=1) is $(d + \frac{d\times(l-1)}{L}) \times \frac{d}{L}$=360 $\times$ 60. Assume we set N=3 for depthwise graph convolutions and M=6 for layerwise graph convolutions. We first use layerwise graph convolutions by dividing the input into 6 groups, where each one has the size $\frac{d}{M}$=60. Then we feed the first group to the first layer. Next we use depthwise graph convolutions to further split the input into 3 groups. We now have 3 weight matrices for the first layer, each one with the size $\frac{d\times(2l-1)}{M} \times \frac{d}{M\times N}$= 20 $\times$ 20. With the increase of the feature dimension d and the number of layer L, more prominent parameter efficiency can be observed.

4.5.3 Weight Tied Convolutions

We further adopt a more aggressive strategy where parameters are shared across all layers. This further significantly reduces the size of the model. Theoretically, weight tied networks can be unrolled to any depth, typically with improved feature abstractions as depth increases (Bai, Kolter, and Koltun, 2019a). Recently, weight tied SANs were explored to regularize the training and help with generalization (Dehghani et al., 2019; Lan et al., 2020). Mathematically, graph convolution can be rewritten as:

$$\mathbf{H}_{l+1} = \phi(\mathbf{A}\hat{\mathbf{H}}_l\mathbf{W} + \mathbf{b}), \tag{4.17}$$

where $\mathbf{W}$ and $\mathbf{b}$ are shared parameters for all convolutional layers. To stabilize training, a gating mechanism was introduced to graph neural networks in order to build graph recurrent networks (Li et al., 2016b; Song et al., 2018a), where parameters are shared across states (time steps). However, the graph convolutional structure is very deep (e.g., 36 layers). Instead, we adopt a jumping connection (Xu et al., 2018), which forms the final representation $\mathbf{H}_{final}$ based on the output of all layers. This connection mechanism can be considered deep supervision (Lee et al., 2015; Bai, Kolter, and Koltun, 2019b) for training deep convolutional neural networks. Formally, the $\mathbf{H}_{final}$ of LDGCNs which have L layer is obtained by: $\mathbf{H}_{final} = \mathcal{F}(\hat{\mathbf{H}}_L, ..., \hat{\mathbf{H}}_1)$, where $\mathcal{F}$ is a linear transformation.

4.6 Experiments of LDGCNs

4.6.1 Experimental Setup

We evaluate our model on three datasets including LDC2015E86 (AMR1.0), LDC2017T10 (AMR2.0) and LDC2020T02 (AMR3.0) datasets, which have 16,833, 36,521 and 55,635 instances for training, respectively. Both AMR1.0 and AMR2.0 have 1,368 instances for development, and 1,371 instances for testing. AMR3.0 has 1,722 instances for development and 1,898 instances for testing. Following Zhu et al. (2019), we use byte pair encodings (Sennrich, Haddow, and Birch, 2016) to deal with rare words.

Following Guo et al. (2019), we stack 4 LDGCN blocks as the encoder of our model. Each block consists of two sub-blocks where the bottom one contains 6 layers and the top one contains 3 layers. The hidden dimension of LDGCN model is 480. Other model hyperparameters are set as λ=0.7, K=2 for dynamic fusion mechanism, N=2 for depthwise graph convolutions and M=6 and 3 for layerwise graph convolutions for the bottom and top sub-blocks, respectively. For the decoder, we employ the same attention-based LSTM as in previous work (Beck, Haffari, and Cohn, 2018; Guo et al., 2019; Damonte and Cohen, 2019). Following Wang, Wan, and Jin (2020), we use a transformer as the decoder for large-scale evaluation. For fair comparisons, we use the same optimization and regularization strategies as in Guo et al. (2019). All hyperparameters are tuned on the development set[4].

For evaluation, we report BLEU scores (Papineni et al., 2002), CHRF++ (Popovic, 2017) scores and METEOR scores (Denkowski and Lavie, 2014) with additional human evaluation results.

4.6.2 Main Results

We consider two kinds of baseline models: 1) models based on Recurrent Neural Networks (Konstas et al., 2017; Cao and Clark, 2019) and Graph Neural Networks (GNNs) (Song et al., 2018a; Beck, Haffari, and Cohn, 2018; Damonte and Cohen, 2019; Guo et al., 2019; Ribeiro, Gardent, and Gurevych, 2019). These models use an attention-based LSTM decoder. 2) models based on SANs (Zhu et al., 2019) and structured SANs (Cai and Lam, 2020b; Zhu et al., 2019; Wang, Wan, and Jin, 2020). Specifically, Zhu et al. (2019) leverage additional SANs to incorporate the relational encoding whereas Cai and Lam (2020b) use GRUs. Additional results of ensemble models are also included.

[4]Hyperparameter search; all hyperparameters are attached in the supplementary material.

Model	T	AMR1.0				AMR2.0			
		B	C	M	#P	B	C	M	#P
Seq2Seq (Cao and Clark, 2019)	S	23.5	-	-	-	26.8	-	-	-
GraphLSTM (Song et al., 2018a)	S	23.3	-	-	-	24.9	-	-	-
GGNNs (Beck, Haffari, and Cohn, 2018)	S	-	-	-	-	23.3	50.4	-	28.3M
GCNLSTM (Damonte and Cohen, 2019)	S	24.4	-	23.6	-	24.5	-	24.1	30.8M‡
DCGCN (Guo et al., 2019)	S	25.7	54.5‡	31.5‡	18.6M‡	27.6	57.3	34.0	19.1M
DualGraph (Ribeiro, Gardent, and Gurevych, 2019)	S	24.3	53.8‡	30.5	60.3M‡	27.9	58.0‡	33.2	61.7M‡
Seq2Seq (Konstas et al., 2017)	E	-	-	-	-	26.6	52.5	-	142M
GGNNs (Beck, Haffari, and Cohn, 2018)	E	-	-	-	-	27.5	53.5	-	141M
DCGCN (Guo et al., 2019)	E	-	-	-	-	30.4	59.6	-	92.5M
Transformer (Zhu et al., 2019)	S	25.5	59.9	33.1	49.1M	27.4	61.9	34.6	-
GT_Dual (Wang, Wan, and Jin, 2020)	S	25.9	-	-	19.9M	29.3	59.0	-	19.9M
GT_GRU (Cai and Lam, 2020b)	S	27.4	56.4	32.9	30.8M	29.8	59.4	35.1	32.2M
GT_SAN (Zhu et al., 2019)	S	29.7	60.7‡	35.5	49.3M	31.8	61.8‡	36.4	54.0M‡
LDGCN_WT	S	28.6	58.5	33.1	**10.6M**	31.9	61.2	36.3	**11.8M**
LDGCN_GC	S	**30.8**	**61.8**	**36.4**	12.9M	**33.6**	**63.2**	**37.5**	13.6M

TABLE 4.11: Main results on AMR-to-text generation. S, E, B, C, M and #P denote single, ensemble, BLEU, CHRF++, METEOR and the model size in terms of parameters, respectively. Results with ‡ are obtained from the authors.

The results are reported in Table 4.11. We also conduct the statistical significance tests. All our proposed systems are significant over the baseline at $p < 0.01$, tested by bootstrap resampling (Koehn, 2004). Our model has two variants based on different parameter saving strategies, including LDGCN_WT (weight tied) and LDGCN_GC (group convolutions), and both of them use the dynamic fusion mechanism (DFM).

LDGCN v.s. Structured SANs. Compared to state-of-the-art structured SANs (GT_SAN), the performance of LDGCN_GC is 1.1 and 1.8 BLEU points higher on AMR1.0 and AMR2.0, respectively. Moreover, LDGCN_GC requires only about a quarter of the number of parameters (12.9M vs 49.0M, and 13.6M vs 54.0M). Our more lightweight variant LDGCN_WT achieves better BLEU scores than GT_SAN on AMR2.0 while using only 1/5 of their model parameters. However, LDGCN_WT obtains lower scores on AMR1.0 than GT_SAN. We hypothesize that weight tied convolutions require more data to train as we observe severe oscillations when training the model on the small AMR1.0 dataset. The oscillation is reduced when we train it on the larger AMR2.0 dataset and the semi-supervised dataset.

LDGCN v.s. Other GNNs. Both LDGCN models significantly outperform GNN-based models. For example, LDGCN_GC surpasses DCGCN by 5.1 points on AMR1.0 and surpasses DualGraph by 5.7 points on AMR2.0. Moreover, the single LDGCN model achieves consistently better results than previous ensemble GNN-based models in BLEU, CHRF++ and METEOR scores. In particular, on AMR2.0, LDGCN_WT obtains 1.5 BLEU points higher than the DCGCN ensemble model, while requiring only about 1/8 of the number of parameters. We also evaluate our model on the latest AMR3.0 dataset. Results are shown in Table 4.12. LDGCN_WT and LDGCN_GC consistently outperform GNN-based models including DCGCN and GGNNs on this larger dataset. These results suggest that LDGCN model is able to learn better representation more efficiently.

Model	B	C	M	#P
GGNNs (Beck, Haffari, and Cohn, 2018)	26.7†	57.2†	33.1†	30.9M†
DCGCN (Guo et al., 2019)	29.8‡	59.9‡	35.6‡	22.2M‡
LDGCN_WT	33.0	62.6	36.5	**11.5M**
LDGCN_GC	**34.3**	**63.7**	**38.2**	14.3M

TABLE 4.12: Results on AMR3.0. B, C, M and #P denote BLEU, CHRF++, METEOR and the model size in terms of parameters, respectively. † denotes results based on open implementations and ‡ are obtained from authors.

Model	#P	External	B
Seq2Seq (Konstas et al., 2017)	-	2M	32.3
Seq2Seq (Konstas et al., 2017)	-	20M	33.8
GraphLSTM (Song et al., 2018a)	-	2M	33.6
Transformer (Wang, Wan, and Jin, 2020)	-	2M	35.1
GT_Dual (Wang, Wan, and Jin, 2020)	78.4M	2M	36.4
LDGCN_GC	23.2M	0.5M	36.0
LDGCN_WT	**20.8M**	**0.5M**	**36.8**

TABLE 4.13: Results on AMR1.0 with external training data.

Large-scale Evaluation. We further evaluate LDGCNs on a large-scale dataset. Following Wang, Wan, and Jin (2020), we first use the additional data to pretrain the model, then fine-tune it on the gold data. Evaluation results are reported in Table 4.13. Using 0.5M additional data, LDGCN_WT outperforms all models including structured SANs with 2M additional data. These results show that our model is more effective in terms of using a larger dataset. Interestingly, LDGCN_WT consistently outperforms LDGCN_GC under this setting. Unlike training the model on AMR1.0, training LDGCN_WT on the large-scale dataset has fewer oscillations, which confirms our hypothesis that sufficient data acts as a regularizer to stabilize the training process of weight tied models.

4.6.3 Development Experiments

We conduct an ablation study to demonstrate how dynamic fusion mechanism and parameter saving strategies are beneficial to the lightweight model with better performance based on development of experimental results on AMR1.0. Results are shown in Table 4.14. DeepGCN is the model with dense connections (Huang et al., 2017; Guo et al., 2019). DeepGCN+GC+DF and DeepGCN+WT+DF are essentially LDGCN_GC and LDGCN_WT models in Section 4.6.2, respectively.

Dynamic Fusion Mechanism. The performance of DeepGCN+DF is 1.1 BLEU points higher than DeepGCN, which demonstrates that our dynamic fusion mechanism is beneficial for graph encoding when applied alone. Adding the group graph convolutions strategies gives a BLEU score of 30.3, which is only 0.1 points lower than DeepGCN+DF. This result shows that the representation learning ability of the dynamic fusion mechanism is robust against parameter

Model	#Parameters	BLEU
DeepGCN	19.9M	29.3
DeepGCN+DF	19.9M	30.4
DeepGCN+GC	12.9M	29.0
DeepGCN+GC+DF (LDGCN_GC)	12.9M	30.3
DeepGCN+WT	10.6M	27.4
DeepGCN+WT+DF (LDGCN_WT)	10.6M	28.3

TABLE 4.14: Comparisons between baselines. +DF denotes dynamic fusion mechanism. +WT and +GC refer to weight tied and group convolutions.

Model	Inference Speed
Transformer	1.00x
DeepGCN	1.21x
LDGCN_WT	1.22x
LDGCN_GC	1.17x

TABLE 4.15: Speed comparisons between baselines. For inference speed, the higher the better. Implementations are based on MXNet (Chen et al., 2015) and the Sockeye neural machine translation toolkit (Felix et al., 2017). Results on speed are based on beam size 10, batch size 30 on an NVIDIA RTX 1080 GPU.

sharing and reduction. We also observe that the mechanism helps to alleviate oscillation when training the weight tied model. DeepGCN+WT+DF achieves better results than DeepGCN+WT, which is hard to converge when training it on the small AMR1.0 dataset.

Parameter Saving Strategy. Table 4.14 demonstrates that although the performance of DeepGCN+GC is only 0.3 BLEU points lower than that of DeepGCN, DeepGCN+GC only requires 65% of the number of parameters of DeepGCN. Furthermore, by introducing the dynamic fusion mechanism, the performance of DeepGCN+GC is improved greatly and is in fact on par with DeepGCN. Also, DeepGCN+GC+DF does not rely on any kind of self-attention layers, hence, its number of parameters is much smaller than that of graph transformers, i.e., DeepGCN+GC+DF only needs $1/4$ to $1/3$ the number of parameters of graph transformers, as shown in Table 4.11. On the other hand, DeepGCN+WT is more efficient than DeepGCN+GC. As shown in Table 4.13, with an increase in training data, more prominent parameter efficiency can be observed.

Time Cost Analysis. As shown in the Table 4.15, all three GCN-based models outperform the SAN-based model in terms of speed because the computation of attention weights scales quadratically while convolutions scale linearly with respect to the input graph size. LDGCN_GC is slightly slower than the other two models, since it requires additional tensor split operations. We believe that state-of-the-art structured SANs are also strictly slower than vanilla SANs, as they require additional neural components, such as GRUs, to encode structural information in the AMR graph. In summary, our model not only has better parameter efficiency, but also lower time costs.

Model	Similarity	Readability
DualGraph (Ribeiro, Gardent, and Gurevych, 2019)	65.07	68.78
GT_SAN (Zhu et al., 2019)	69.63	72.23
DeepGCN	68.91	71.45
LDGCN_GC	**71.92**	**74.16**

TABLE 4.16: Human evaluation. We also perform significance tests. Results are statistically significant with $p < 0.05$.

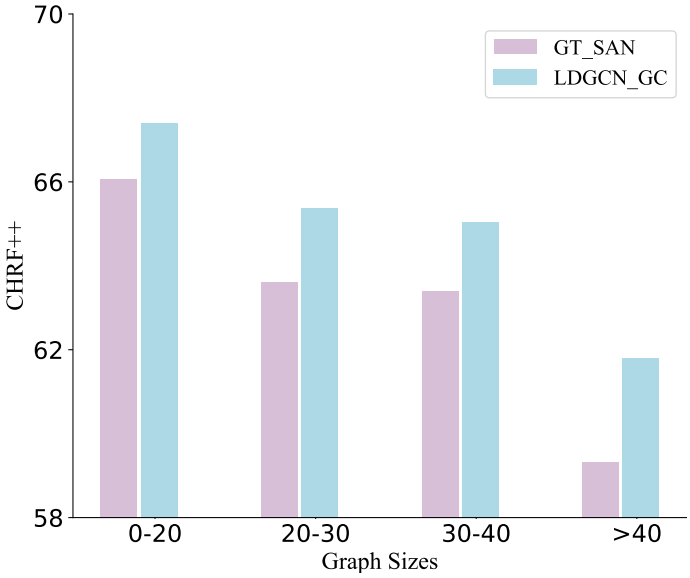


FIGURE 4.11: Performance against graph sizes.

4.6.4 Human Evaluation

We further assess the quality of the generated sentences with human evaluation. Following Ribeiro, Gardent, and Gurevych (2019), two evaluation criteria are used: (i) meaning similarity: how close in meaning the generated text is to the gold sentence; (ii) readability: how well the generated sentence reads. We randomly select 100 sentences generated by 4 models. 30 human subjects rate the sentences on a 0-100 rating scale. The evaluation is conducted separately and subjects were first given brief instructions explaining the criteria of assessment. For each sentence, we collect scores from 5 subjects and average them. Models are ranked according to the mean of sentence-level scores. Also, we apply a quality control step filtering subjects who do not score some faked and known sentences properly.

As shown in Table 4.16, LDGCN_GC has better human rankings in terms of both meaning similarity and readability than the state-of-the-art GNN-based (DualGraph) and SAN-based model (GT_SAN). DeepGCN without the dynamic fusion mechanism obtains lower scores than GT_SAN, which further confirms that synthesizing higher order information helps in learning better graph representations.

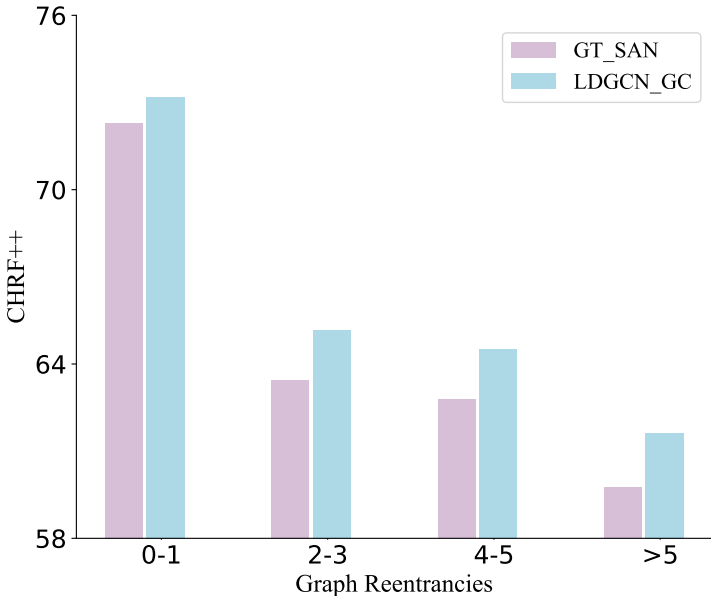


FIGURE 4.12: Performance against graph re-entrancies.

4.6.5 Additional Analysis

To further reveal the source of performance gains, we perform additional analysis based on the characteristics of AMR graphs, i.e., graph size and graph reentrancy (Damonte and Cohen, 2019). All experiments are conducted on the AMR2.0 test set and CHRF++ scores are reported.

Graph Size. As shown in Figure 4.11, the size of AMR graphs is partitioned into four categories ((0, 20], (20, 30], (30, 40], $>$ 40), Overall, LDGCN_GC outperforms the best-reported GT_SAN model across all graph sizes, and the performance gap becomes more profound with the increase of graph sizes. Although both models have sharp performance degradation for extremely large graphs ($>$ 40), the performance of LDGCN_GC is more stable. Such a result suggests that our model can better deal with large graphs with more complicated structures.

Graph re-entrancies. Reentrancies describe the co-references and control structures in AMR graphs. A graph is considered more complex if it contains more re-entrancies. In Figure 4.12, we show how the LDGCN_GC and GT_SAN generalize to different scales of reentrancies. Again, LDGCN_GC consistently outperforms GT_SAN and the performance gap becomes noticeably wider when the number of re-entrancies increases. These results suggest that our model can better model the complex dependencies in AMR graphs.

Case Study. Table 4.17 shows the generated sentence of an AMR graph from four models together with the gold reference. The phrase "trust me" is the beginning of the sentence. DualGraph fails to decode it. On the other hand, GT_SAN successfully generates the second half of the sentence, i.e., "rather than let them get even worse", but it fails to capture the meaning of word "early" in its output, which is a critical part. DeepGCN parses both "early" and "get even worse" in the results. However, the readability of the generated sentence is not satisfactory. Compared to baselines, LDGCN is able to produce the best result, which has a correct starting

(m / multi-sentence :snt1 (t / trust-01 :ARG2 (i / i)) :snt2 (g / good-02 :ARG1 (g2 / **get-01** :ARG1 (t2 / thing :mod (t3 / this)) :time e.10,12 (e / **early** :degree (m2 / most) :compared-to (p / possible-01 :ARG1 g2)) :ARG1-of (i2 / instead-of-91 :ARG2 (l / let-01 :ARG1 (w / **worsen-01** :ARG1 t2 :mod (e2 / **even**))))) :degree e.5 (m3 / more)))
Reference: trust me , it 's better to get these things as early as possible rather than let them get even worse .
DualGraph: so to me , this is the best thing to get these things as they can , instead of letting it even worse .
DeepGCN: i trust me , it 's better that these things get in the early than letting them even get worse .
GT_SAN: trust me , this is better to get these things , rather than let it even get worse .
LDGCN_GC: trust me . better to get these things as early as possible , rather than letting them even make worse .

TABLE 4.17: Example outputs.

phrase and captures the semantic meaning of critical words such as "early" and "get even worse" while also attaining good readability.

Chapter 5

Modelling Dependency Graph for Relation Extraction

5.1 Introduction

In this chapter, we mainly focus on integrating graph structures for better relation extraction. As discussed in Section 2.3.4, there exist two challenges in developing graph-based models.

The first one is how to maintain a balance between including and excluding information associated with the dependency graph. Existing approaches leverage rule-based pruning strategies, which might eliminate some important information for correct prediction. Ideally, the model should be able to learn how to keep relevant information and remove irrelevant information. In Section 4.3, we propose the novel Attention Guided Graph Convolutional Networks (AGGCNs) , which operate directly on the full dependency graph. Intuitively, we develop a "soft pruning" strategy that transforms the original dependency tree into a fully connected edge-weighted graph. These weights can be viewed as the strength of relatedness between nodes, which can be learned in an end-to-end fashion by using self-attention mechanism (Vaswani et al., 2017). Experiments show that the AGGCN model is able to achieve better performance for various tasks. For the cross-sentence relation extraction task, our model surpasses the current state-of-the-art models on multi-class ternary and binary relation extraction by 8% and 6% in terms of accuracy respectively. For the large-scale sentence-level extraction task (TACRED; Zhang et al. 2017b), our model is also consistently better than others, showing the effectiveness of the model on a large training set.

The second challenge lies in document-level and bio-medical relation extraction is the lack of accurate dependency graph. For document-level relation extraction, prior efforts construct the document-level graph by co-references and rules. This document-level graph may not be able to capture the complex interactions among mentions and entities. Unlike previous methods, we propose to treat the document-level graph as a latent variable and induce it in an end-to-end fashion. In Section 5.5, we introduce the model that is able to generate document-level dependency structures for capturing non-local interactions between entities. We further develop an iterative refinement strategy, which enables our model to dynamically build the latent structure based on the last iteration, allowing the model to incrementally capture the complex interactions for better multi-hop reasoning (Welbl, Stenetorp, and Riedel, 2018). Experiments show that our model significantly outperforms the existing approaches on DocRED (Yao et al., 2019), a large-scale document-level relation extraction dataset with a large number of entities and relations.

For bio-medical relation extraction, previous research efforts (Peng et al., 2017a; Linfeng et al., 2018) rely on dependency parsers trained on newswire text. In Section 5.7, we introduce a model based on the mechanism of structured attention (Kim et al., 2017; Liu and Lapata, 2018).

Using a variant of the Matrix-Tree Theorem (Tutte, 1984; Koo et al., 2007; Smith and Smith, 2007), our model is able to generate non-projective dependency structure for capturing non-local interactions between entities without recourse to any pre-trained parsers or tree supervision. We further construct multiple forests by projecting the representations of nodes to different representation subspaces, allowing an induction of a more informative latent structure for better relation prediction. We name our proposed model as LF-GCN, where LF is the abbreviation of latent forests. Experiments show that our LF-GCN model is able to achieve better performance on various relation extraction tasks. For the sentence-level tasks, our model surpasses the current stat-of-the-art models on the CPR dataset (Krallinger, Rabal, Akhondi, et al., 2017) and the PGR dataset (Sousa, Lamúrias, and Couto, 2019) by 3.2% and 2.6% in terms of $F1$ score, respectively. For the cross-sentence tasks (Peng et al., 2017a), our model is also consistently better than others, showing its effectiveness on long medical text.

5.2 Related Work

5.2.1 Sentence-Level Relation Extraction.

Early research efforts are based on statistical methods. Tree-based kernels (Zelenko, Aone, and Richardella, 2002) and dependency path-based kernels (Bunescu and Mooney, 2005) are explored to extract the relation. McDonald et al. (2005) construct maximal cliques of entities to predict relations. Mintz et al. (2009) include syntactic features to a statistical classifier. Recently, sequence-based models leverages different neural networks to extract relations, including convolutional neural networks (Zeng et al., 2014; Nguyen and Grishman, 2015; Wang et al., 2016b), recurrent neural networks (Zhou et al., 2016b; Zhang et al., 2017b), the combination of both (Vu et al., 2016) and transformer (Verga, Strubell, and McCallum, 2018).

Graph-based approaches also try to incorporate structural information into the neural models. Peng et al. (2017a) first split the dependency graph into two DAGs, then extend the tree LSTM model (Tai, Socher, and Manning, 2015) over these two graphs for n-ary relation extraction. Closest to our work, Linfeng et al. (2018) use graph recurrent networks (Song et al., 2018a) to directly encode the whole dependency graph without breaking it. The contrast between our model and theirs is reminiscent of the contrast between CNN and RNN. Various pruning strategies have also been proposed to distill the dependency information in order to further improve the performance. Xu et al. (2015b) and Xu et al. (2015c) adapt neural models to encode the shortest dependency path. Miwa and Bansal (2016) apply LSTM model over the LCA subtree of two entities. Liu et al. (2015) combine the shortest dependency path and the dependency subtree. Zhang, Qi, and Manning (2018) adopt a path-centric pruning strategy. Unlike these strategies that remove edges in preprocessing, our model learns to assign each edge a different weight in an end-to-end fashion.

5.2.2 Document-Level Relation Extraction.

Early efforts focus on predicting relations between entities within a single sentence by modeling interactions in the input sequence or the corresponding dependency tree. These approaches do not consider interactions across mentions and ignore relations expressed across sentence boundaries. However, valuable relational information between entities is expressed by multiple mentions across sentence boundaries in real-world scenarios (Peng et al., 2017a). Therefore, the scope of extraction has recently been expanded to cross-sentence level (Quirk and Poon, 2017;

Gupta et al., 2018; Linfeng et al., 2018; Song et al., 2019a). Instead of using discourse structure understanding techniques (Liu, Cohen, and Lapata, 2018; Liu, Cohen, and Lapata, 2019), these approaches leverage the dependency graph to capture inter-sentence interactions, and their scope is still limited to several sentences. More recently, the extraction scope has been expanded to the entire document (Verga, Strubell, and McCallum, 2018; Jia, Wong, and Poon, 2019; Sahu et al., 2019; Christopoulou, Miwa, and Ananiadou, 2019) in the biomedical domain by only considering a few relations among chemicals. Unlike previous work, we focus on document-level relation extraction datasets (Yao et al., 2019; Li et al., 2016a; Wu et al., 2019b) from different domains with a large number of relations and entities, which require understanding a document and performing multi-hop reasoning.

5.2.3 Bio-Medical Relation Extraction.

In the bio-medical domain, McDonald et al. (2005) extract n-ary relations by first factoring the n-ary relation into pair-wise relations between all entity pairs, and then constructing maximal cliques of related entities. Recent efforts leverage different neural networks to predict relations between entities by modeling interactions in the 1-best dependency tree, including graph LSTM (Peng et al., 2017a) and graph recurrent networks (Linfeng et al., 2018). Instead of using the 1-best dependency tree, dependency forests are used to alleviate the error cascading caused by an out-of-domain parser. Song et al. (2019a) construct a dependency forest by adding additional edges with high confidence scores given by a dependency parser trained on the news domain (Dozat and Manning, 2017) or merging the K-best trees (Eisner, 1996) by combining identical dependency edges. Lifeng et al. (2020) construct full forests represented by a 3-dimensional tensor generated by a pre-trained parser fine-tuned by the relation prediction loss. The dependency parser can be adjusted with the loss from the end task.

5.3 Attention Guided Graph for Relation Extraction

In this section, we will present the basic components used for constructing our Attention Guided Graph Convolutional Networks (AGGCNs). The AGGCN model is composed of M identical blocks as shown in Figure 5.1. Each block consists of three types of layers: attention guided layer, densely connected layer and linear combination layer. We first introduce the attention guided layer of the AGGCN model.

5.3.1 Attention Guided Layer

As we discussed in Section 2.3.1, most existing pruning strategies are predefined. They prune the full tree into a subtree, based on which the adjacency matrix is constructed. In fact, such strategies can also be viewed as a form of hard attention (Xu et al., 2015a), where edges that connect nodes not on the resulting subtree will be directly assigned zero weights (not attended). Such strategies might eliminate relevant information from the original dependency tree. Instead of using rule-based pruning, we develop a "soft pruning" strategy in the attention guided layer, which assigns weights to all edges. These weights can be learned by the model in an end-to-end fashion. The attention guided layer is based on graph convolutions, so we will first introduce the graph convolutional networks.

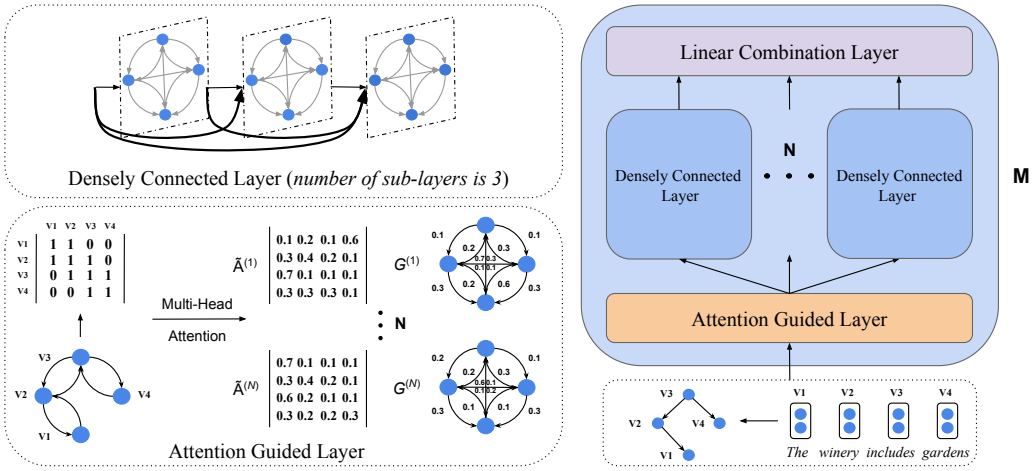


FIGURE 5.1: The AGGCN model is shown with an example sentence and its dependency tree. It is composed of M identical blocks and each block has three types of layers as shown on the right.

Graph Convolutional Networks GCNs are neural networks that operate directly on graph structures (Kipf and Welling, 2017). Here we mathematically illustrate how multi-layer GCNs work on a graph. Given a graph with n nodes, we can represent the graph with an $n \times n$ adjacency matrix $\mathbf{A}$. Marcheggiani and Titov (2017) extend GCNs for encoding dependency trees by incorporating directionality of edges into the model. They add a self-loop for each node in the tree. Opposite direction of a dependency arc is also included, which means $\mathbf{A}_{ij} = 1$ and $\mathbf{A}_{ji} = 1$ if there is an edge going from node i to node j, otherwise $\mathbf{A}_{ij} = 0$ and $\mathbf{A}_{ji} = 0$. The convolution computation for node i at the l-th layer, which takes the input feature representation $\mathbf{h}^{(l-1)}$ as input and outputs the induced representation $\mathbf{h}_i^{(l)}$, can be defined as:

$$\mathbf{h}_i^{(l)} = \rho\Big(\sum_{j=1}^{n} \mathbf{A}_{ij}\mathbf{W}^{(l)}\mathbf{h}_j^{(l-1)} + \mathbf{b}^{(l)}\Big) \tag{5.1}$$

where $\mathbf{W}^{(l)}$ is the weight matrix, $\mathbf{b}^{(l)}$ is the bias vector, and ρ is an activation function (e.g., RELU). $\mathbf{h}_i^{(0)}$ is the initial input $\mathbf{x}_i$, where $\mathbf{x}_i \in \mathbb{R}^d$ and d is the input feature dimension.

Attention Guided Graph In order to construct the attention guided graph, we transform the original dependency tree into a fully connected edge-weighted graph by constructing an attention guided adjacency matrix $\tilde{\mathbf{A}}$. Each $\tilde{\mathbf{A}}$ corresponds to a certain fully connected graph and each entry $\tilde{\mathbf{A}}_{ij}$ is the weight of the edge going from node i to node j. As shown in Figure 5.1, $\tilde{\mathbf{A}}^{(1)}$ represents a fully connected graph $G^{(1)}$. $\tilde{\mathbf{A}}$ can be constructed by using self-attention mechanism (Cheng, Dong, and Lapata, 2016), which is an attention mechanism (Bahdanau, Cho, and Bengio, 2015) that captures the interactions between two arbitrary positions of a single sequence. Once we get $\tilde{\mathbf{A}}$, we can use it as the input for the computation of the later graph convolutional layer. Note that the size of $\tilde{\mathbf{A}}$ is the same as the original adjacency matrix $\mathbf{A}$ ($n \times n$). Therefore, no additional computational overhead is involved. The key idea behind the attention guided layer is to use attention for inducing relations between nodes, especially for

those connected by indirect, multi-hop paths. These soft relations can be captured by differentiable functions in the model.

Here we compute $\tilde{\mathbf{A}}$ by using multi-head attention (Vaswani et al., 2017), which allows the model to jointly attend to information from different representation subspaces. The calculation involves a query and a set of key-value pairs. The output is computed as a weighted sum of the values, where the weight is computed by a function of the query with the corresponding key.

$$\tilde{\mathbf{A}}^{(\mathbf{t})} = softmax(\frac{Q\mathbf{W}_i^Q \times (K\mathbf{W}_i^K)^T}{\sqrt{d}}) \tag{5.2}$$

where Q and K are both equal to the collective representation $\mathbf{h}^{(l-1)}$ at layer $l-1$ of the AGGCN model. The projections are parameter matrices $\mathbf{W}_i^Q \in \mathbb{R}^{d\times d}$ and $\mathbf{W}_i^K \in \mathbb{R}^{d\times d}$. $\tilde{\mathbf{A}}^{(\mathbf{t})}$ is the t-th attention guided adjacency matrix corresponding to the t-th head. Up to N matrices are constructed, where N is a hyper-parameter.

Figure 5.1 shows an example that the original adjacency matrix is transformed into multiple attention guided adjacency matrices. Accordingly, the input dependency tree is converted into multiple fully connected edge-weighted graphs. In practice, we treat the original adjacency matrix as an initialization so that the dependency information can be captured in the node representations for later attention calculation. The attention guided layer is included starting from the second block.

5.3.2 Densely Connected Layer

Unlike previous pruning strategies, which lead to a resulting structure that is smaller than the original structure, our attention guided layer outputs a larger fully connected graph. Following Guo et al. (2019), we introduce dense connections (Huang et al., 2017) into the AGGCN model in order to capture more structural information on large graphs. With the help of dense connections, we are able to train a deeper model, allowing rich local and non-local information to be captured for learning a better graph representation.

Dense connectivity is shown in Figure 5.1. Direct connections are introduced from any layer to all its preceding layers. Mathematically, we first define $\mathbf{g}_j^{(l)}$ as the concatenation of the initial node representation and the node representations produced in layers $1, \cdots, l-1$:

$$\mathbf{g}_j^{(l)} = [\mathbf{x}_j; \mathbf{h}_j^{(1)}; ...; \mathbf{h}_j^{(l-1)}] \tag{5.3}$$

In practice, each densely connected layer has L sub-layers. The dimensions of these sub-layers d_{hidden} are decided by L and the input feature dimension d. In AGGCNs, we use $d_{hidden} = d/L$. For example, if the densely connected layer has 3 sub-layers and the input dimension is 300, the hidden dimension of each sub-layer will be $d_{hidden} = d/L = 300/3 = 100$. Then we concatenate the output of each sub-layer to form the new representation. Therefore, the output dimension is 300 (3×100). Different from the GCN model whose hidden dimension is larger than or equal to the input dimension, the AGGCN model shrinks the hidden dimension as the number of layers increases in order to improve the parameter efficiency similar to DenseNets (Huang et al., 2017).

Since we have N different attention guided adjacency matrices, N separate densely connected layers are required. Accordingly, we modify the computation of each layer as follows

(for the t-th matrix $\tilde{\mathbf{A}}^{(\mathbf{t})}$):

$$\mathbf{h}_{t_i}^{(l)} = \rho\Big(\sum_{j=1}^{n} \tilde{\mathbf{A}}_{ij}^{(t)} \mathbf{W}_t^{(l)} \mathbf{g}_j^{(l)} + \mathbf{b}_t^{(l)}\Big) \tag{5.4}$$

where $t = 1, ..., N$ and t selects the weight matrix and bias term associated with the attention guided adjacency matrix $\tilde{\mathbf{A}}^{(\mathbf{t})}$. The column dimension of the weight matrix increases by d_{hidden} per sub-layer, i.e., $\mathbf{W}_t^{(l)} \in \mathbb{R}^{d_{hidden} \times d^{(l)}}$, where $d^{(l)} = d + d_{hidden} \times (l-1)$.

5.3.3 Linear Combination Layer

The AGGCN model includes a linear combination layer to integrate representations from N different densely connected layers. Formally, the output of the linear combination layer is defined as:

$$\mathbf{h}_{comb} = \mathbf{W}_{comb}\mathbf{h}_{out} + \mathbf{b}_{comb} \tag{5.5}$$

where $\mathbf{h}_{out}$ is the output by concatenating outputs from N separate densely connected layers, i.e., $\mathbf{h}_{out} = [\mathbf{h}^{(1)}; ...; \mathbf{h}^{(N)}] \in \mathbb{R}^{d \times N}$. $\mathbf{W}_{comb} \in \mathbb{R}^{(d \times N) \times d}$ is a weight matrix and $\mathbf{b}_{comb}$ is a bias vector for the linear transformation.

In summary, the AGGCN model is composed of M identical blocks and each block has three types of layers as shown in Figure 5.1. Every block takes node embeddings and adjacency matrix that represents the graph as inputs. Then N attention guided adjacency matrices are constructed by using multi-head attention as shown at bottom left. The original dependency tree is transformed into N different fully connected edge-weighted graphs (self-loops are omitted for simplification). Numbers near the edges represent the weights in the matrix. Resulting matrices are fed into N separate densely connected layers, generating new representations. Top left shows an example of the densely connected layer, where the number (L) of sub-layers is 3 (L is a hyper-parameter). Each sub-layer concatenates all preceding outputs as the input. Eventually, a linear combination is applied to combine outputs of N densely connected layers into hidden representations.

5.3.4 Relation Classifier

After applying the AGGCN model over the dependency tree, we obtain hidden representations of all tokens. Given these representations, the goal of relation extraction is to predict a relation among entities. Following Zhang, Qi, and Manning (2018), we concatenate the sentence representation and entity representations to get the final representation for classification. First we need to obtain the sentence representation h_{sent}. It can be computed as:

$$h_{sent} = f(\mathbf{h_{mask}}) = f(\text{AGGCN}(\mathbf{x})) \tag{5.6}$$

where $\mathbf{h_{mask}}$ represents the masked collective hidden representations. Masked here means we only select representations of tokens that are not entity tokens in the sentence. $f : \mathbb{R}^{d \times n} \rightarrow \mathbb{R}^{d \times 1}$ is a max pooling function that maps from n output vectors to 1 sentence vector. Similarly, we can obtain entity representations. For the i-th entity, its representation h_{e_i} can be computed as:

$$h_{e_i} = f(\mathbf{h_{e_i}}) \tag{5.7}$$

where $\mathbf{h_{e_i}}$ indicates the hidden representation corresponding to the i-th entity.[1] Entity representations will be concatenated with sentence representation to form a new representation. Then we apply a feed-forward neural network (FFNN) over the concatenated representations inspired by relational reasoning works (Santoro et al., 2017; Lee et al., 2017):

$$h_{final} = \text{FFNN}([h_{sent}; h_{e_1}; ...h_{e_i}]) \tag{5.8}$$

where h_{final} will be taken as inputs to a logistic regression classifier to make a prediction.

5.4 Experiments of Attention Guided Graph

5.4.1 Data

We evaluate the performance of our model on two tasks, namely, cross-sentence n-ary relation extraction and sentence-level relation extraction.

For the cross-sentence n-ary relation extraction task, we use the dataset introduced in (Peng et al., 2017a), which contains 6,987 ternary relation instances and 6,087 binary relation instances extracted from PubMed.[2] Most instances contain multiple sentences and each instance is assigned with one of the five labels, including: "resistance or nonresponse", "sensitivity", "response", "resistance" and "None". We consider two specific tasks for evaluation, i,e., binary-class n-ary relation extraction and multi-class n-ary relation extraction. For binary-class n-ary relation extraction, we follow Peng et al. (2017a) to binarize multi-class labels by grouping the four relation classes as "Yes" and treating "None" as "No".

For the sentence-level relation extraction task, we follow the experimental settings in Zhang, Qi, and Manning (2018) to evaluate our model on the TACRED dataset (Zhang et al., 2017b) and Semeval-10 Task 8 (Hendrickx et al., 2010). With over 106K instances, the TACRED dataset introduces 41 relation types and a special "no relation" type to describe the relations between the mention pairs in instances. Subject mentions are categorized into person and organization, while object mentions are categorized into 16 fine-grained types, including date, location, etc. Semeval-10 Task 8 is a public dataset, which contains 10,717 instances with 9 relations and a special "other" class.

5.4.2 Experimental Setup

We tune the hyper-parameters according to results on the development sets. For the cross-sentence n-ary relation extraction task, we use the same data split used in Linfeng et al. (2018)[2], while for the sentence-level relation extraction task, we use the same development set from Zhang, Qi, and Manning (2018)[3].

We choose the number of heads N for attention guided layer from $\{1, 2, 3, 4\}$, the block number M from $\{1, 2, 3\}$, the number of sub-layers L in each densely connected layer from $\{2, 3, 4, 5, 6\}$. Through preliminary experiments on the development sets, we find that the combinations (N=2, M=2, L=5, d_{hidden}=340) and (N=3, M=2, L=5, d_{hidden}=300) give the best

[1] The number of entities is fixed in n-ary relation extraction task. It is 3 for the first dataset and 2 for the second.

[2] The dataset is available at https://github.com/freesunshine0316/nary-grn

[3] https://nlp.stanford.edu/projects/tacred/

Model	**Binary-class**				**Multi-class**	
	T		B		T	B
	`Single`	`Cross`	`Single`	`Cross`	`Cross`	`Cross`
Feature-Based (Quirk and Poon, 2017)	74.7	77.7	73.9	75.2	-	-
SPTree (Miwa and Bansal, 2016)	-	-	75.9	75.9	-	-
Graph LSTM-EMBED (Peng et al., 2017a)	76.5	80.6	74.3	76.5	-	-
Graph LSTM-FULL (Peng et al., 2017a)	77.9	80.7	75.6	76.7	-	-
+ multi-task	-	82.0	-	78.5	-	-
Bidir DAG LSTM (Linfeng et al., 2018)	75.6	77.3	76.9	76.4	51.7	50.7
GS GLSTM (Linfeng et al., 2018)	80.3	83.2	83.5	83.6	71.7	71.7
GCN (Full Tree) (Zhang, Qi, and Manning, 2018)	84.3	84.8	84.2	83.6	77.5	74.3
GCN (K=0) (Zhang, Qi, and Manning, 2018)	85.8	85.8	82.8	82.7	75.6	72.3
GCN (K=1) (Zhang, Qi, and Manning, 2018)	85.4	85.7	83.5	83.4	78.1	73.6
GCN (K=2) (Zhang, Qi, and Manning, 2018)	84.7	85.0	83.8	83.7	77.9	73.1
AGGCN (ours)	**87.1**	**87.0**	**85.2**	**85.6**	**79.7**	**77.4**

TABLE 5.1: Average test accuracies in five-fold validation for binary-class n-ary relation extraction and multi-class n-ary relation extraction. "T" and "B" denote ternary drug-gene-mutation interactions and binary drug-mutation interactions, respectively. `Single` means that we report the accuracy on instances within single sentences, while `Cross` means the accuracy on all instances. K in the GCN models means that the preprocessed pruned trees include tokens up to distance K away from the dependency path in the LCA subtree.

results on cross-sentence n-ary relation extraction and sentence-level relation extraction, respectively. GloVe (Pennington, Socher, and Manning, 2014)[4] vectors are used as the initialization for word embeddings.

Models are evaluated using the same metrics as previous work (Linfeng et al., 2018; Zhang, Qi, and Manning, 2018). We report the test accuracy averaged over five cross validation folds (Linfeng et al., 2018) for the cross-sentence n-ary relation extraction task. For the sentence-level relation extraction task, we report the micro-averaged $F1$ scores for the TACRED dataset and the macro-averaged $F1$ scores for the SemEval dataset (Zhang, Qi, and Manning, 2018). For TACRED dataset, we report the mean test $F1$ score by using 5 models from independent runs.

5.4.3 Results on Cross-Sentence n-ary Relation Extraction

For cross-sentence n-ary relation extraction task, we consider three kinds of models as baselines: 1) a feature-based classifier (Quirk and Poon, 2017) based on shortest dependency paths between all entity pairs, 2) Graph-structured LSTM methods, including Graph LSTM (Peng et al., 2017a), bi-directional DAG LSTM (Bidir DAG LSTM; Linfeng et al. 2018) and Graph State LSTM (GS GLSTM; Linfeng et al. 2018). These methods extend LSTM to encode graphs constructed from input sentences with dependency edges, 3) Graph convolutional networks (GCN) with pruned trees, which have shown efficacy on the relation extraction task (Zhang, Qi, and Manning, 2018)[5]. Additionally, we follow Linfeng et al. (2018) to consider the tree-structured

[4]We use the 300-dimensional Glove word vectors trained on the Common Crawl corpus https://nlp.stanford.edu/projects/glove/

[5]The results are produced by the open implementation of Zhang, Qi, and Manning (2018).

Model	P	R	$F1$
LR (Zhang et al., 2017b)	**73.5**	49.9	59.4
SDP-LSTM (Xu et al., 2015c)*	66.3	52.7	58.7
Tree-LSTM (Tai, Socher, and Manning, 2015)**	66.0	59.2	62.4
PA-LSTM (Zhang et al., 2017b)	65.7	**64.5**	65.1
GCN (Zhang, Qi, and Manning, 2018)	69.8	59.0	64.0
C-GCN (Zhang, Qi, and Manning, 2018)	69.9	63.3	66.4
AGGCN (ours)	69.9	60.9	65.1
C-AGGCN (ours)	73.1	64.2	**69.0**

TABLE 5.2: Results on the TACRED dataset. Model with * indicates that the results are reported in Zhang et al. (2017b), while model with ** indicates the results are reported in Zhang, Qi, and Manning (2018).

LSTM method (SPTree; Miwa and Bansal 2016) on drug-mutation binary relation extraction. Main results are shown in Table 5.1.

We first focus on the binary-class n-ary relation extraction task. For ternary relation extraction (first two columns in Table 5.1), our AGGCN model achieves accuracies of 87.1 and 87.0 on instances within single sentence (`Single`) and on all instances (`Cross`), respectively, which outperform all the baselines. More specifically, our AGGCN model surpasses the state-of-the-art Graph-structured LSTM model (GS GLSTM) by 6.8 and 3.8 points for the `Single` and `Cross` settings, respectively. Compared to GCN models , our model obtains 1.3 and 1.2 points higher than the best performing model with pruned tree (K=1). For binary relation extraction (third and fourth columns in Table 5.1), AGGCN consistently outperforms GS GLSTM and GCN as well.

These results suggest that, compared to previous full tree based methods, e.g., GS GLSTM, AGGCN is able to extract more information from the underlying graph structure to learn a more expressive representation through graph convolutions. AGGCN also performs better than GCNs, although its performance can be boosted via pruned trees. We believe this is because of the combination of densely connected layer and attention guided layer. The dense connections could facilitate information propagation in large graphs, enabling AGGCN to efficiently learn from long-distance dependencies without pruning techniques. Meanwhile, the attention guided layer can further distill relevant information and filter out noises from the representation learned by the densely connected layer.

We next show the results on the multi-class classification task (last two columns in Table 5.1). We follow Linfeng et al. (2018) to evaluate our model on all instances for both ternary and binary relations. This fine-grained classification task is much harder than the coarse-grained classification task. As a result, the performance of all models degrades a lot. However, our model still obtains 8.0 and 5.7 points higher than GS GLSTM for ternary and binary relations, respectively. We also notice that our AGGCN achieves a better test accuracy than all GCN models, which further demonstrates its ability to learn better representations from full trees.

Model	$F1$
SVM (Rink and Harabagiu, 2010)	82.2
SDP-LSTM (Xu et al., 2015c)	83.7
SPTree (Miwa and Bansal, 2016)	84.4
PA-LSTM (Zhang et al., 2017b)	82.7
C-GCN (Zhang, Qi, and Manning, 2018)	84.8
C-AGGCN (ours)	**85.7**

TABLE 5.3: Results on the SemEval dataset.

5.4.4 Results on Sentence-Level Relation Extraction

We now report the results on the TACRED dataset for the sentence-level relation extraction task in Table 5.2. We compare our model against two kinds of models: 1) dependency-based models, 2) sequence-based models. Dependency-based models include the logistic regression classifier (LR; Zhang et al. 2017b), Shortest Path LSTM (SDP-LSTM) (Xu et al., 2015c), Tree-structured neural model (Tree-LSTM; Tai, Socher, and Manning 2015), GCN and Contextualized GCN (C-GCN; Zhang, Qi, and Manning 2018). Both GCN and C-GCN models use the pruned trees. For sequence-based models, we consider the state-of-the-art Position Aware LSTM (PA-LSTM; Zhang et al. 2017b).

As shown in Table 5.2, the logistic regression classifier (LR) obtains the highest precision score. We hypothesize that the reason behind this is due to the data imbalance. This feature-based method tends to predict the relation to be the highly frequent labels (e.g., "per:title"). Therefore, it has a high precision while has a relatively low recall. On the other hand, neural models achieve a better balance between precision and recall.

Since GCN and C-GCN already show their superiority over other dependency-based models and PA-LSTM, we mainly compare our AGGCN model with them. We can observe that AGGCN outperforms GCN by 1.1 $F1$ points. We speculate that the limited improvement is due to the lack of contextual information about word order or disambiguation. Similar to C-GCN (Zhang, Qi, and Manning, 2018), we extend our AGGCN model with a bi-directional LSTM network to capture the contextual representations which are subsequently fed into AGGCN layers. We term the modified model as C-AGGCN. Our C-AGGCN model achieves an $F1$ score of 68.2, which outperforms the state-of-art C-GCN model by 1.8 points. We also notice that AGGCN and C-AGGCN achieve better precision and recall scores than GCN and C-GCN, respectively. The performance gap between GCNs with pruned trees and AGGCNs with full trees empirically show that the AGGCN model is better at distinguishing relevant information from irrelevant information for learning a better graph representation.

We also evaluate our model on the SemEval dataset under the same settings as in Zhang, Qi, and Manning (2018). Results are shown in Table 5.3. This dataset is much smaller than TACRED (only 1/10 of TACRED in terms of the number of instances). Our C-AGGCN model (85.7) consistently outperforms the C-GCN model (84.8), showing good generalizability.

5.4.5 Analysis and Discussion

Ablation Study. We examine the contributions of two main components, namely, densely connected layers and attention guided layers, using the best-performing C-AGGCN model on

Model	$F1$
C-AGGCN	69.0
– Attention-guided layer (AG)	67.1
– Dense connected layer (DC)	67.3
– AG, DC	66.7
– Feed-Forward layer (FF)	67.8

TABLE 5.4: An ablation study for C-AGGCN model.

Model	$F1$
C-AGGCN (Full tree)	69.0
C-AGGCN (K=2)	67.5
C-AGGCN (K=1)	67.9
C-AGGCN (K=0)	67.0

TABLE 5.5: Results of C-AGGCN with pruned trees.

the TACRED dataset. Table 5.4 shows the results. We can observe that adding either attention guided layers or densely connected layers improves the performance of the model. This suggests that both layers can assist GCNs to learn better information aggregations, producing better representations for graphs, where the attention-guided layer seems to be playing a more significant role. We also notice that the feed-forward layer is effective in our model. Without the feed-forward layer, the result drops to an $F1$ score of 67.8.

Performance with Pruned Trees. Table 5.5 shows the performance of the C-AGGCN model with pruned trees, where K means that the pruned trees include tokens that are up to distance K away from the dependency path in the LCA subtree. We can observe that all the C-AGGCN models with varied values of K are able to outperform the state-of-the-art C-GCN model (Zhang, Qi, and Manning, 2018) (reported in Table 5.2). Specifically, with the same setting as K=1, C-AGGCN surpasses C-GCN by 1.5 points of $F1$ score. This demonstrates that, with the combination of densely connected layer and attention guided layer, C-AGGCN can learn better representations of graphs than C-GCN for downstream tasks. In addition, we notice that the performance of C-AGGCN with full trees outperforms all C-AGGCNs with pruned trees. These results further show the superiority of "soft pruning" strategy over hard pruning strategy in utilizing full tree information.

Performance against Sentence Length. Figure 5.2 shows the $F1$ scores of three models under different sentence lengths. We partition the sentence length into five classes ($<$ 20, [20, 30), [30, 40), [40, 50), $\geq$50). In general, C-AGGCN with full trees outperforms C-AGGCN with pruned trees and C-GCN against various sentence lengths. We also notice that C-AGGCN with pruned trees performs better than C-GCN in most cases. Moreover, the improvement achieved by C-AGGCN with pruned trees decays when the sentence length increases. Such a performance degradation can be avoided by using full trees, which provide more information of the underlying graph structures. Intuitively, with the increase of the sentence length, the

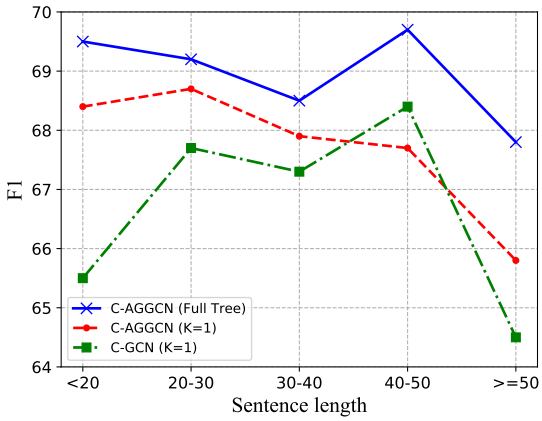


FIGURE 5.2: Comparison of models against different sentence lengths.

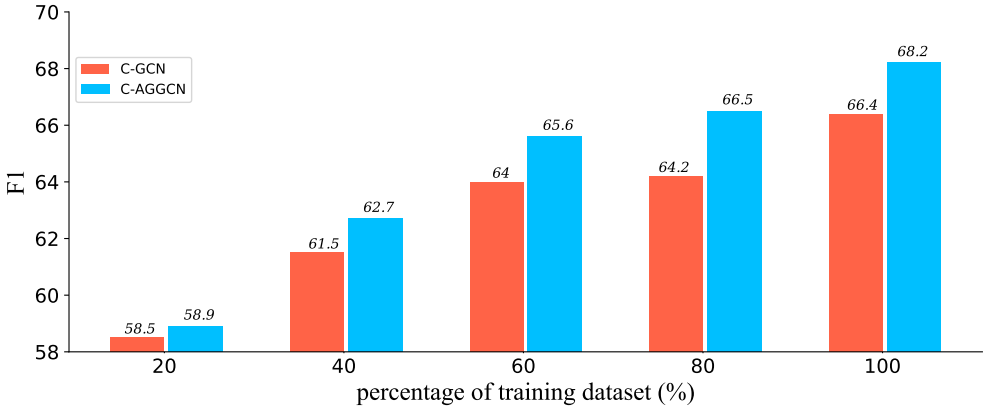


FIGURE 5.3: Comparison of models against different training data sizes.

dependency graph becomes larger as more nodes are included. This suggests that C-AGGCN can benefit more from larger graphs (full tree).

Performance against Training Data Size. Figure 5.3 shows the performance of C-AGGCN and C-GCN against different training settings. We consider five training settings (20%, 40%, 60%, 80%, 100% of the training data). C-AGGCN consistently outperforms C-GCN under the same amount of training data. When the size of training data increases, we can observe that the performance gap becomes more obvious. Particularly, using 80% of the training data, the C-AGGCN model is able to achieve a $F1$ score of 66.5, higher than C-GCN trained on the whole dataset. These results demonstrate that our model is more effective in terms of using training resources.

5.5 Latent Graph Refinement for Document-Level Relation Extraction

In this section, we present our proposed Latent Structure Refinement (LSR) model for the document-level relation extraction task. Our LSR model consists of three components: node constructor, dynamic reasoner, and relation classifier. The node constructor first encodes each sentence of an input document and outputs contextual representations. Representations that correspond to mentions and tokens on the shortest dependency path in a sentence are extracted as nodes. The dynamic reasoner is then applied to induce a document-level structure based on the extracted nodes. Representations of nodes are updated based on information propagation on the latent structure, which is iteratively refined. Final representations of nodes are used to calculate classification scores by the classifier.

5.5.1 Node Constructor

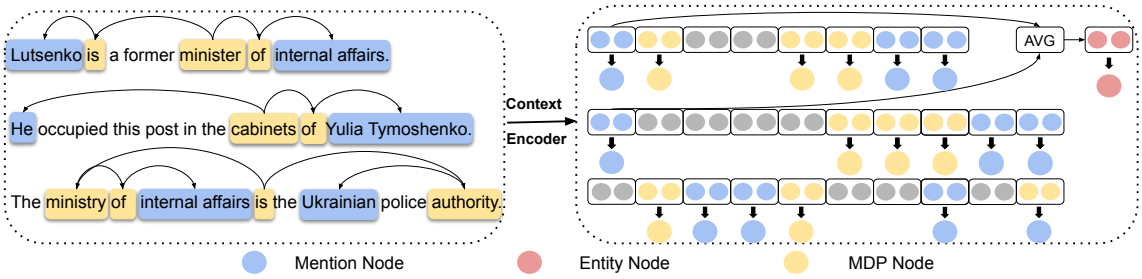


FIGURE 5.4: Overview of the Node Constructor.

Node constructor encodes sentences in a document into contextual representations and constructs representations of mention nodes, entity nodes and meta dependency paths (MDP) nodes. As shown in Figure 5.4, A context encoder is applied to get the contextualized representations of sentences. The representations of mentions and words in the meta dependency paths are extracted as mention nodes and MDP nodes. An average pooling is used to construct the entity node from the mention nodes. For example, the entity node *Lutsenko* is constructed by averaging representations of its mentions *Lutsenko* and *He*.

Context Encoding Given a document $\boldsymbol{d}$, each sentence d_i in it is fed to the context encoder, which outputs the contextualized representations of each word in d_i. The context encoder can be a bi-directional LSTM (BiLSTM; Schuster and Paliwal 1997 or BERT (Devlin et al., 2019). Here we use the BiLSTM as an example:

$$\overleftarrow{\mathbf{h}}^i_j = \mathbf{LSTM}_l(\overleftarrow{\mathbf{h}}^i{}_{j+1}, \gamma^i_j) \tag{5.9}$$

$$\overrightarrow{\mathbf{h}}^i_j = \mathbf{LSTM}_r(\overrightarrow{\mathbf{h}}^i{}_{j-1}, \gamma^i_j) \tag{5.10}$$

where $\overleftarrow{\mathbf{h}}^i_j$, $\overleftarrow{\mathbf{h}}^i{}_{j+1}$, $\overrightarrow{\mathbf{h}}^i_j$ and $\overrightarrow{\mathbf{h}}^i{}_{j-1}$ represent the hidden representations of the j-th, (j+1)-th and (j-1)-th token in the sentence d_i of two directions, and γ^i_j denotes the word embedding of the j-th

token. Contextual representation of each token in the sentence is represented as $\mathbf{h}_j^i = [\overleftarrow{\mathbf{h}_j^i}; \overrightarrow{\mathbf{h}_j^i}]$ by concatenating hidden states of two directions, where $\mathbf{h}_j^i \in \mathbb{R}^d$ and d is the dimension.

Node Extraction We construct three types of nodes for a document-level graph: mention nodes, entity nodes and meta dependency paths (MDP) nodes as shown in Figure 5.4. Mention nodes correspond to different mentions of entities in each sentence. The representation of an entity node is computed as the average of its mentions. To build a document-level graph, existing approaches use all nodes in the dependency tree of a sentence (Sahu et al., 2019) or one sentence-level node by averaging all token representations of the sentence (Christopoulou, Miwa, and Ananiadou, 2019). Alternatively, we use tokens on the shortest dependency path between mentions in the sentence. The shortest dependency path has been widely used in sentence-level relation extraction as it is able to effectively make use of relevant information while ignoring irrelevant information (Bunescu and Mooney, 2005; Xu et al., 2015b; Xu et al., 2015c). Unlike sentence-level extraction, where each sentence only has two entities, each sentence here may involve multiple mentions.

5.5.2 Dynamic Reasoner

The dynamic reasoner has two modules, structure induction and multi-hop reasoning as shown in Figure 5.5. The structure induction module is used to learn a latent structure of a document-level graph. The multi-hop reasoning module is used to perform inference on the induced latent structure, where representations of each node will be updated based on the information aggregation scheme. We stack N blocks in order to iteratively refine the latent document-level graph for better reasoning.

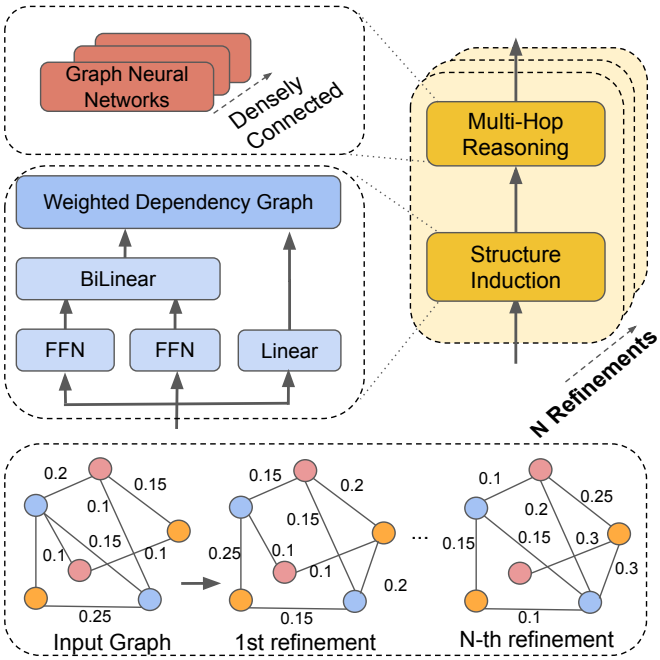


FIGURE 5.5: Overview of the Dynamic Reasoner.

Structure Induction Unlike existing models that use co-reference links (Sahu et al., 2019) or heuristics (Christopoulou, Miwa, and Ananiadou, 2019) to construct a document-level graph for reasoning, our model treats the graph as a latent variable and induces it in an end-to-end fashion. The structure induction module is built based on the structured attention (Kim et al., 2017; Liu and Lapata, 2018). Inspired by Liu and Lapata (2018), we use a variant of Kirchhoff's Matrix-Tree Theorem (Tutte, 1984; Koo et al., 2007; Smith and Smith, 2007) to induce the latent dependency structure.

Let $\mathbf{u}_i$ denote the contextual representation of the i-th node, where $\mathbf{u}_i \in \mathbb{R}^d$, we first calculate the pair-wise unnormalized attention score $\mathbf{s}_{ij}$ between the i-th and the j-th node with the node representations $\mathbf{u}_i$ and $\mathbf{u}_j$. The score $\mathbf{s}_{ij}$ is calculated by two feed-forward neural networks and a bilinear transformation:

$$\mathbf{s}_{ij} = (\tanh(\mathbf{W}_p\mathbf{u}_i))^T\mathbf{W}_b(\tanh(\mathbf{W}_c\mathbf{u}_j)) \tag{5.11}$$

where $\mathbf{W}_p \in \mathbb{R}^{d\times d}$ and $\mathbf{W}_c \in \mathbb{R}^{d\times d}$ are weights for two feed-forward neural networks, d is the dimension of the node representations, and $\tanh$ is applied as the activation function. $\mathbf{W}_b \in \mathbb{R}^{d\times d}$ are the weights for the bilinear transformation. Next we compute the root score $\mathbf{s}_i^r$ which represents the unnormalized probability of the i-th node to be selected as the root node of the structure:

$$\mathbf{s}_i^r = \mathbf{W}_r\mathbf{u}_i \tag{5.12}$$

where $\mathbf{W}_r \in \mathbb{R}^{1\times d}$ is the weight for the linear transformation. Following Koo et al. (2007) and Smith and Smith (2007), we calculate the marginal probability of each dependency edge of the document-level graph. For a graph $\mathbf{G}$ with n nodes, we first assign non-negative weights $\mathbf{P} \in \mathbb{R}^{n\times n}$ to the edges of the graph:

$$\mathbf{P}_{ij} = \begin{cases} 0 & \text{if } i = j \\ \exp(\mathbf{s}_{ij}) & \text{otherwise} \end{cases} \tag{5.13}$$

where $\mathbf{P}_{ij}$ is the weight of the edge between the i-th and the j-th node. We then define the Laplacian matrix $\mathbf{L} \in \mathbb{R}^{n\times n}$ of $\mathbf{G}$ in Equation 5.14, and its variant $\hat{\mathbf{L}} \in \mathbb{R}^{n\times n}$ in Equation 5.15 for further computations (Koo et al., 2007; Smith and Smith, 2007).

$$\mathbf{L}_{ij} = \begin{cases} \sum_{i'=1}^{n} \mathbf{P}_{i'j} & \text{if } i = j \\ -\mathbf{P}_{ij} & \text{otherwise} \end{cases} \tag{5.14}$$

$$\hat{\mathbf{L}}_{ij} = \begin{cases} \exp(\mathbf{s}_i^r) & \text{if } i = 1 \\ \mathbf{L}_{ij} & \text{if } i > 1 \end{cases} \tag{5.15}$$

We use $\mathbf{A}_{ij}$ to denote the marginal probability of the dependency edge between the i-th and the j-th node. Then, $\mathbf{A}_{ij}$ can be derived based on Equation 5.16, where δ is the Kronecker delta.

$$\begin{aligned}\mathbf{A}_{ij} = &(1-\delta_{1,j})\mathbf{P}_{ij}[\hat{\mathbf{L}}^{-1}]_{ij} \\ &-(1-\delta_{i,1})\mathbf{P}_{ij}[\hat{\mathbf{L}}^{-1}]_{ji}\end{aligned} \tag{5.16}$$

where $\mathbf{L} \in \mathbb{R}^{n\times n}$ is the Laplacian matrix for graph $\mathbf{G}$ and $\hat{\mathbf{L}} \in \mathbb{R}^{n\times n}$ is a variant of $\mathbf{L}$ that takes the root node into consideration, and δ is the Kronecker delta. $\mathbf{A}_{ij}$ is the marginal probability of the dependency edge between the i-th and j-th words. Here, $\mathbf{A} \in \mathbb{R}^{n\times n}$ can be interpreted as a weighted adjacency matrix of the document-level entity graph. Finally, we can feed $\mathbf{A} \in \mathbb{R}^{n\times n}$ into the multi-hop reasoning module to update the representations of nodes in the latent structure.

Multi-hop Reasoning Graph neural networks have been widely used in different tasks to perform multi-hop reasoning (Song et al., 2018b; Yang et al., 2019; Tu et al., 2019; Lin et al., 2019), as they are able to effectively collect relevant evidence based on an information aggregation scheme. Specifically, our model is based on graph convolutional networks (GCNs) (Kipf and Welling, 2017) to perform reasoning.

Formally, given a graph $\mathbf{G}$ with n nodes, which can be represented with an $n \times n$ adjacency matrix $\mathbf{A}$ induced by the previous structure induction module, the convolution computation for the node i at the l-th layer, which takes the representation $\mathbf{u}_i^{l-1}$ from previous layer as input and outputs the updated representations $\mathbf{u}_i^l$, can be defined as:

$$\mathbf{u}_i^l = \sigma(\sum_{j=1}^{n} \mathbf{A}_{ij}\mathbf{W}^l\mathbf{u}_i^{l-1} + \mathbf{b}^l) \tag{5.17}$$

where $\mathbf{W}^l$ and $\mathbf{b}^l$ are the weight matrix and bias vector for the l-th layer, respectively. σ is the ReLU (Nair and Hinton, 2010) activation function. $\mathbf{u}_i^0 \in \mathbb{R}^d$ is the initial contextual representation of the i-th node constructed by the node constructor.

Following Guo et al. (2019), we use dense connections to the GCNs in order to capture more structural information on a large document-level graph. With the help of dense connections, we are able to train a deeper model, allowing richer local and non-local information to be captured for learning a better graph representation. The computations on each graph convolution layer is similar to Equation 5.17.

Mathematically, we first define $\mathbf{g}_i^l$ as the concatenation of the initial node representations and the node representations produced in layers $1, ..., l-1$:

$$\mathbf{g}_i^l = [\mathbf{g}_i^0; \mathbf{g}_i^1; ..., \mathbf{g}_i^{l-1}] \tag{5.18}$$

The computations on each graph convolution layer are performed based on Equation 5.17.

$$\mathbf{u}_i^l = \sigma(\sum_{j=1}^{n} \mathbf{A}_{ij}\mathbf{W}^l\mathbf{g}_i^{l-1} + \mathbf{b}^l) \tag{5.19}$$

Though structured attention (Kim et al., 2017; Liu and Lapata, 2018) is able to automatically induce a latent structure, recent research efforts show that the induced structure is relatively shallow and may not be able to model the complex dependencies for document-level input (Liu,

Titov, and Lapata, 2019; Ferracane et al., 2019). Unlike previous work (Liu and Lapata, 2018) that only induces the latent structure once, we repeatedly refine the document-level graph based on the updated representations, allowing the model to infer a more informative structure that goes beyond simple parent-child relations.

As shown in Figure 5.5, we stack N blocks of the dynamic reasoner in order to induce the document-level structure N times. Intuitively, the reasoner induces a shallow structure at early iterations since the information propagates mostly between neighboring nodes. As the structure gets more refined by interactions with richer non-local information, the induction module is able to generate a more informative structure.

5.5.3 Relation Classifier

After N times of refinement, we obtain representations of all the nodes. Following Yao et al. (2019), for each entity pair $(\mathbf{e}_i, \mathbf{e}_j)$, we use a bilinear function to compute the probability for each relation type r as:

$$P(r|\mathbf{e}_i, \mathbf{e}_j) = \sigma(\mathbf{e}_i^T \mathbf{W}_\mathbf{e} \mathbf{e}_j + \mathbf{b}_e)_r \tag{5.20}$$

where $\mathbf{W}_e \in \mathbb{R}^{d \times k \times d}$ and $\mathbf{b}_e \in \mathbb{R}^k$ are trainable weights and bias, with k being the number of relation categories, σ is the *sigmoid* function, and the subscript r in the right side of the equation refers to the relation type.

5.6 Experiments of Latent Graph Refinement

5.6.1 Data

We evaluate our model on DocRED (Yao et al., 2019), the largest human-annotated dataset for document-level relation extraction, and another two popular document-level relation extraction datasets in the biomedical domain, including Chemical-Disease Reactions (CDR; Li et al. 2016a) and Gene-Disease Associations (GDA; Wu et al. 2019b). DocRED contains $3,053$ documents for training, $1,000$ for development and $1,000$ for test, totally with $132,375$ entities and $56,354$ relational facts. CDR consists of 500 training instances, 500 development instances, and 500 testing instances. GDA contains $29,192$ documents for training and $1,000$ for test. We follow (Christopoulou, Miwa, and Ananiadou, 2019) to split the training set of GDA into an 80/20 split for training and development.

With more than 40% of the relational facts requiring reading and reasoning over multiple sentences, DocRED significantly differs from previous sentence-level datasets (Doddington et al., 2004; Hendrickx et al., 2010; Zhang et al., 2017b). Unlike existing document-level datasets (Li et al., 2016a; Quirk and Poon, 2017; Peng et al., 2017a; Verga, Strubell, and McCallum, 2018; Jia, Wong, and Poon, 2019) that are in the specific biomedical domain considering only the drug-gene-disease relation, DocRED covers a broad range of categories with 96 relation types.

Batch size	20
Learning rate	0.001
Optimizer	Adam
Hidden size	120
Induction block number	2
GCN dropout	0.3

TABLE 5.6: Hyper-parameters of LSR.

5.6.2 Experimental Setup

We use spaCy[6] to get the meta dependency paths of sentences in a document. Following Yao et al. (2019) and Wang et al. (2019), we use the GloVe (Pennington, Socher, and Manning, 2014) embedding with BiLSTM, and Uncased BERT-Base (Devlin et al., 2019) as the context encoder. All hyper-parameters are tuned based on the development set. We list some of the important hyper-parameters in Table 5.6.

Following Yao et al. (2019), we use F_1 and Ign F_1 as the evaluation metrics. Ign F_1 denotes F_1 scores excluding relational facts shared by the training and dev/test sets. F_1 scores for intra- and inter-sentence entity pairs are also reported. Evaluation on the test set is done through CodaLab[7].

5.6.3 Main Results

We compare our proposed LSR with the following three types of competitive models on the DocRED dataset, and show the main results in Table 5.7.

- **Sequence-based Models.** These models leverage different neural architectures to encode sentences in the document, including convolutional neural networks (CNN; Zeng et al. 2014), LSTM, bi-directional LSTM (BiLSTM; Cai, Zhang, and Wang 2016 and attention-based LSTM (ContextAware; Sorokin and Gurevych 2017).

- **Graph-based Models.** These models construct task-specific graphs for inference. GCNN (Sahu et al., 2019) constructs a document-level graph by co-reference links, and then applies relational GCNs for reasoning. EoG (Christopoulou, Miwa, and Ananiadou, 2019) is the state-of-the-art document-level relation extraction model in the biomedical domain. EoG first uses heuristics to construct the graph, then leverages an edge-oriented model to perform inference. GCNN and EoG are based on static structures. GAT (Veličković et al., 2018) is able to learn the weighted graph structure based on a local attention mechanism. AGGCN (Guo, Zhang, and Lu, 2019) is the state-of-the-art sentence-level relation extraction model, which constructs the latent structure by self-attention. These two models are able to dynamically construct task-specific structures.

- **BERT-based Models.** These models fine-tune BERT (Devlin et al., 2019) for DocRED. Specifically, Two-Phase BERT (Wang et al., 2019) is the best reported model. It is a pipeline model, which predicts if the relation exists between entity pairs in the first phase and predicts the type of the relation in the second phase.

[6] https://spacy.io/
[7] https://competitions.codalab.org/competitions/20717

	Dev				Test	
Model	Ign $F1$	$F1$	Intra-$F1$	Inter-$F1$	Ign $F1$	$F1$
CNN (Yao et al., 2019)	41.58	43.45	51.87*	37.58*	40.33	42.26
LSTM (Yao et al., 2019)	48.44	50.68	56.57*	41.47*	47.71	50.07
BiLSTM (Yao et al., 2019)	48.87	50.94	57.05*	43.49*	48.78	51.06
ContexAware (Yao et al., 2019)	**48.94**	51.09	56.74*	42.26*	48.40	50.70
GCNN† (Sahu et al., 2019)	46.22	51.52	57.78	44.11	49.59	51.62
EoG† (Christopoulou, Miwa, and Ananiadou, 2019)	45.94	52.15	58.90	44.60	49.48	51.82
GAT† (Veličković et al., 2018)	45.17	51.44	58.14	43.94	47.36	49.51
AGGCN† (Guo, Zhang, and Lu, 2019)	46.29	52.47	58.76	45.45	48.89	51.45
GloVe+LSR	48.82	**55.17**	**60.83**	**48.35**	**52.15**	**54.18**
BERT (Wang et al., 2019)	-	54.16	61.61*	47.15*	-	53.20
Two-Phase BERT (Wang et al., 2019)	-	54.42	61.80*	47.28*	-	53.92
BERT+LSR	**52.43**	**59.00**	**65.26**	**52.05**	**56.97**	**59.05**

TABLE 5.7: Main results on DocRED. † denotes models adapted to DocRED based on open implementations. $*$ denotes results based on re-trained models.

As shown in Table 5.7, LSR with GloVe achieves 54.18 F_1, which is the new state-of-the-art result for models with GloVe. In particular, our model consistently outperforms sequence-based models by a significant margin. For example, LSR improves upon the best sequence-based model BiLSTM by 3.1 points in terms of F_1. This suggests that models which directly encode the entire document are unable to capture the inter-sentence relations present in documents.

Under the same setting, LSR consistently outperforms graph-based models based on static graphs or attention mechanisms. Compared with EoG, LSR achieves 3.0 and 2.4 higher F_1 on development and test set, respectively. We also have similar observations for the GCNN model, which shows that a static document-level graph may not be able to capture the complex interactions in a document. The dynamic latent structure induced by LSR captures richer non-local dependencies. Moreover, LSR also outperforms GAT and AGGCN. This empirically shows that compared to the models that use local attention and self-attention (Veličković et al., 2018; Guo, Zhang, and Lu, 2019), LSR can induce more informative document-level structures for better reasoning. Our LSR model also shows its superiority under the setting of Ign F_1.

In addition, LSR with GloVe obtains better results than two BERT-based models. This empirically shows that our model is able to capture long-range dependencies even without using powerful context encoders. Following Wang et al. (2019), we leverage BERT as the context encoder. As shown in Table 5.7, our LSR model with BERT achieves a 59.05 F_1 score on DocRED, which is a new state-of-the-art result. As of the ACL deadline on the 9th of December 2019, we held the first position on the CodaLab scoreboard under the alias *diskorak*.

Intra- and Inter-sentence Performance The DocRED requires reasoning across sentences for relation prediction.To comply with the nature of the document-level relation extraction task, we analyze intra- and inter-sentence performance on the development set. An entity pair requires inter-sentence reasoning if the two entities from the same document have no mentions in the same sentence. In DocRED's development set, about 45% of entity pairs require information aggregation over multiple sentences.

Under the same setting, our LSR model outperforms all other models in both intra- and inter-sentence settings. The differences in F_1 scores between LSR and other models in the inter-sentence setting tend to be larger than the differences in the intra-sentence setting. These results

demonstrate that the majority of LSR's superiority comes from the inter-sentence relational facts, suggesting that the latent structure induced by our model is indeed capable of synthesizing the information across multiple sentences of a document.

Furthermore, LSR with GloVe also proves better in the inter-sentence setting compared with two BERT-based (Wang et al., 2019) models, indicating latent structure's superiority in resolving long-range dependencies across the whole document compared with the BERT encoder.

Model	$F1$	Intra-$F1$	Inter-$F1$
Gu et al. (2017)	61.3	57.2	11.7
Nguyen and Verspoor (2018)	62.3	-	-
Verga, Strubell, and McCallum (2018)	62.1	-	-
Sahu et al. (2019)	58.6	-	-
Christopoulou, Miwa, and Ananiadou (2019)	63.6	68.2	50.9
LSR	61.2	66.2	50.3
LSR w/o MDP Nodes	**64.8**	**68.9**	**53.1**
Peng, Wei, and Lu (2016)	63.1	-	-
Li et al. (2016c)	67.3	58.9	-
Panyam et al. (2018)	60.3	65.1	45.7
Zheng et al. (2018)	61.5	-	-

TABLE 5.8: Results on the test set of the CDR dataset. Models below the double line use additional training data and/or incorporate external tools.

Model	$F1$	Intra-$F1$	Inter-$F1$
NoInf (Christopoulou, Miwa, and Ananiadou, 2019)	74.6	79.1	49.3
Full (Christopoulou, Miwa, and Ananiadou, 2019)	80.8	84.1	**54.7**
EoG (Christopoulou, Miwa, and Ananiadou, 2019)	81.5	85.2	50.0
LSR	79.6	83.1	49.6
LSR w/o MDP Nodes	**82.2**	**85.4**	51.1

TABLE 5.9: Results on the test set of the GDA dataset.

Results on the Bio-medical Datasets Table 5.8 depicts the comparisons with state-of-the-art models on the CDR dataset. Gu et al. (2017), Nguyen and Verspoor (2018), and Verga, Strubell, and McCallum (2018) leverage sequence-based models. Convolutional neural networks and self-attention networks are used as the encoders. Sahu et al. (2019) and Christopoulou, Miwa, and Ananiadou (2019) use graph-based models. As shown in Table 5.8, our LSR performs worse than the state-of-the-art models. It is challenging for an off-the-shelf parser to get high quality dependency trees in the biomedical domain, as we observe that the MDP nodes extracted by the spaCy parser from the CDR dataset contains much less informative context compared with the nodes from DocRED. Here we introduce a simplified LSR model indicated as "LSR w/o MDP Nodes" , which removes the MDP nodes and builds a fully-connected graph using all tokens of a document. It shows that "LSR w/o MDP Nodes" consistently outperforms sequence-based and graph-based models, indicating the effectiveness of the latent structure. Moreover, the simplified LSR outperforms most of the models with external resources, except for Li et al. (2016c), which

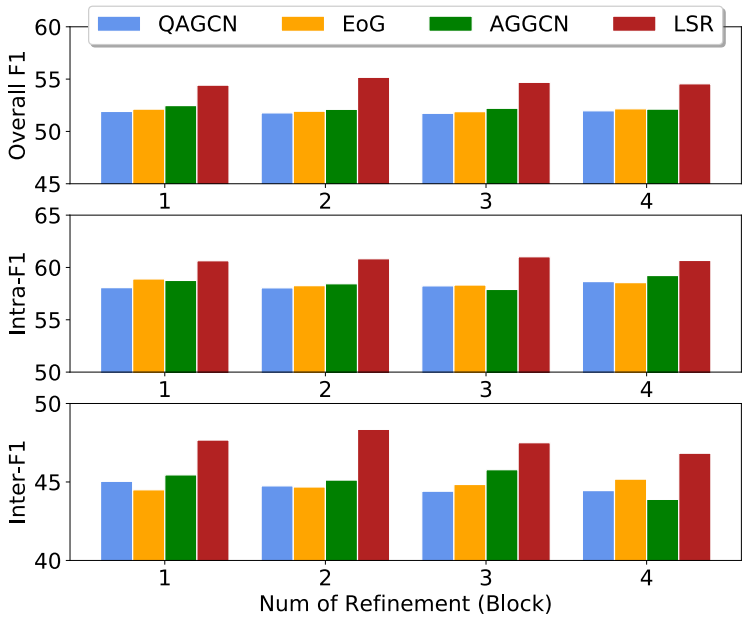


FIGURE 5.6: Performance of different graph structures in QAGCN, EoG, AGGCN and LSR. The number of refinements is ranging from 1 to 4.

leverages co-training with additional unlabeled training data. We believe such a setting also benefits our LSR model.

Table 5.9 shows the results on the distantly supervised GDA dataset. Here "Full" indicates EoG model with a fully connected graph as the inputs, while "NoInf" is a variant of EoG model without inference component (Christopoulou, Miwa, and Ananiadou, 2018). The simplified LSR model achieves the new state-of-the-art result on GDA. The "Full" model (Christopoulou, Miwa, and Ananiadou, 2019) yields a higher $F1$ score on the inter-sentence setting while having a relatively low score on the intra-sentence. It is likely because that this model neglects the differences between relations expressed within the sentence and across sentences.

5.6.4 Model Analysis

In this subsection, we use the development set of DocRED to demonstrate the effectiveness of the latent structure and refinements.

Does Latent Structure Matter? We investigate the extent to which the latent structures, that are induced and iteratively refined by the proposed dynamic reasoner, help to improve the overall performance. We experiment with the three different structures defined below. For fair comparisons, we use the same GCN model to perform multi-hop reasoning for all these structures.

Rule-based Structure: We use the rule-based structure in EoG (Christopoulou, Miwa, and Ananiadou, 2019). Also, We adapt rules from De Cao, Aziz, and Titov (2019) for multi-hop question answering, i.e., each mention node is connected to its entity node and to the same mention nodes across sentences, while mention nodes and MDP nodes which reside in the same sentence are fully connected. The model is termed QAGCN.

Model	$F1$	Intra-$F1$	Inter-$F1$
Full model	55.17	60.83	48.35
- 1 Refinement	54.42	60.46	47.67
- 2 Structure Induction	51.91	58.08	45.04
- 1 Multi-hop Reasoning	54.49	59.75	47.49
- 2 Multi-hop Reasoning	54.24	60.58	47.15
- MDP nodes	54.20	60.54	47.12

TABLE 5.10: Ablation study of LSR on DocRED.

Attention-based Structure: This structure is induced by AGGCN (Guo, Zhang, and Lu, 2019) with multi-head attention (Vaswani et al., 2017). We extend the model from sentence-level to document-level. We explore multiple settings of these models with different block numbers ranging from 1 to 4, where a block is composed of a graph construction component and a densely connected GCN component. As shown in Figure 5.6, LSR outperforms QAGCN, EoG and AGGCN in terms of overall F_1. This empirically confirms our hypothesis that the latent structure induced by LSR is able to capture a more informative context for the entire document.

Does Refinement Matter? As shown in Figure 5.6, our LSR yields the best performance in the second refinement, outperforming the first induction by 0.72% in terms of overall F_1. This indicates that the proposed LSR is able to induce more accurate structures by iterative refinement. However, too many iterations may lead to an F_1 drop due to over-fitting.

Ablation Study Table 5.10 shows F_1 scores of the full LSR model and with different components turned off one at a time. We observe that most of the components contribute to the main model, as the performance deteriorates with any of the components missing. The most significant difference is visible in the structure induction module. Removal of the structure induction part leads to a 3.26 drop in terms of F_1 score. This result indicates that the latent structure plays a key role in the overall performance.

5.6.5 Case Study

In Figure 5.7, we present a case study to analyze why the latent structure induced by our proposed LSR performs better than the structures learned by AGGCN. We visualize the reasoning process for predicting the relation of an entity pair ⟨*Japan*, *World War II*⟩ by LSR and AGGCN in two refinement steps, using the attention scores of the mention *World War II* in each step. We scale all attention scores by 1000 to illustrate them more clearly. As shown in Figure 5.7, in the first refinement of LSR, *Word War II* interacts with several local mentions with higher attention scores, e.g., *0.43* for the mention *Lake Force*, which will be used as a bridge between the mention *Japan* and *World War II*. In the second refinement, the attention scores of several non-local mentions, such as *Japan* and *Imperial Japanese Army*, significantly increase from *0.09* to *0.41*, and *0.17* to *0.37*, respectively, indicating that information is propagated globally at this step. With such intra- and inter-sentence structures, the relation of the entity pair ⟨*Japan*, *World War II*⟩ can be predicted as *"participant of"*, which is denoted by *P1344*. Compared with LSR, the attention scores learned by AGGCN are much more balanced, indicating that the model may not

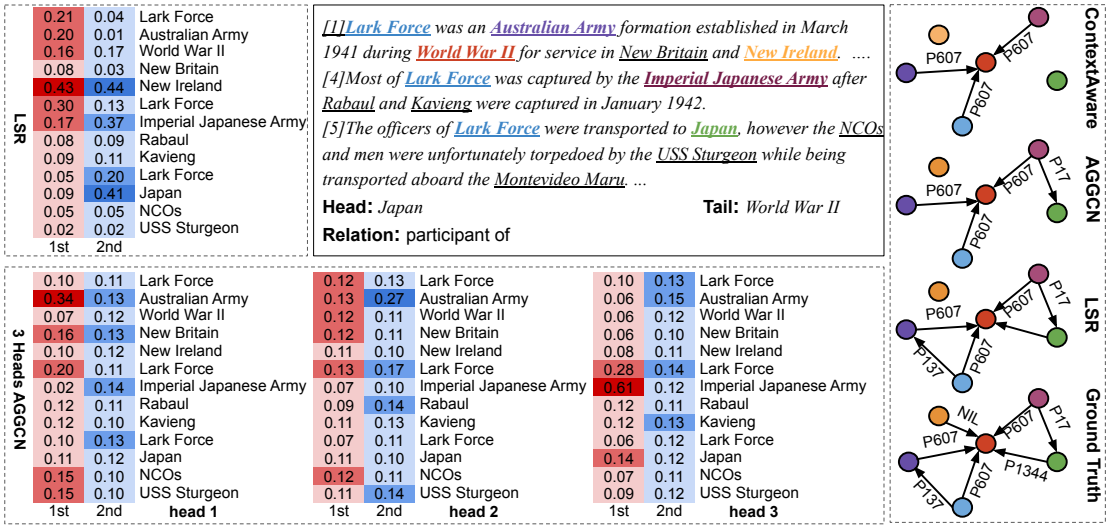


FIGURE 5.7: Case study of an example from the development set of DocRED. Some sentences are omitted due to space limitation.

be able to construct an informative structure for inference, e.g., the highest score is *0.27* in the second head, and most of the scores are near *0.11*.

We also depict the predicted relations of ContextAware, AGGCN and LSR on the graph on the right side of the Figure 5.7. Interested readers could refer to Yao et al. (2019) for the definition of a relation, such as *P607*, *P17*, *etc.* The LSR model proves capable of filling out the missing relation for ⟨*Japan*, *World War II*⟩ that requires reasoning across sentences. However, LSR also attends to the mention *New Ireland* with a high score, thus failing to predict that the entity pair ⟨*New Ireland*, *World War II*⟩ actually has no relation (*NIL* type).

5.7 Modelling Latent Forests for Bio-Medical Relation Extraction

In this section, we introduce our proposed model LF-GCN, which leverages latent forests for bio-medical relation extraction. Existing approaches leverage a dependency parser trained on newswire text (Song et al., 2019a) or a fine-tuned parser for the medical domain (Lifeng et al., 2020) to generate a dependency forest, which is a fully-connected weighted graph. Unlike previous efforts, we treat the forest as a latent variable, which can be learned from a targeting dataset in an end-to-end manner. Inspired by Kim et al. (2017) and Liu and Lapata (2018), we use a variant of Kirchhoff's Matrix-Tree Theorem (Tutte, 1984; Koo et al., 2007; Smith and Smith, 2007) to induce the latent structure of an input sentence. Such latent structure can be viewed as multiple full dependency forests, which efficiently represent all possible dependency trees within a compact and dense structure.

As shown Figure 5.8, LF-GCN is composed of M identical blocks and each block has two components—forest inducer and forest encoder. The forest inducer consists of two sub-modules, where the first sub-module computes N attention matrices based on the multi-head attention, and the second sub-module takes the N attention matrices as inputs to obtain N dependency forests based on the Matrix-Tree Theorem. Then, the forest encoder uses graph neural networks to encode the induced forests.

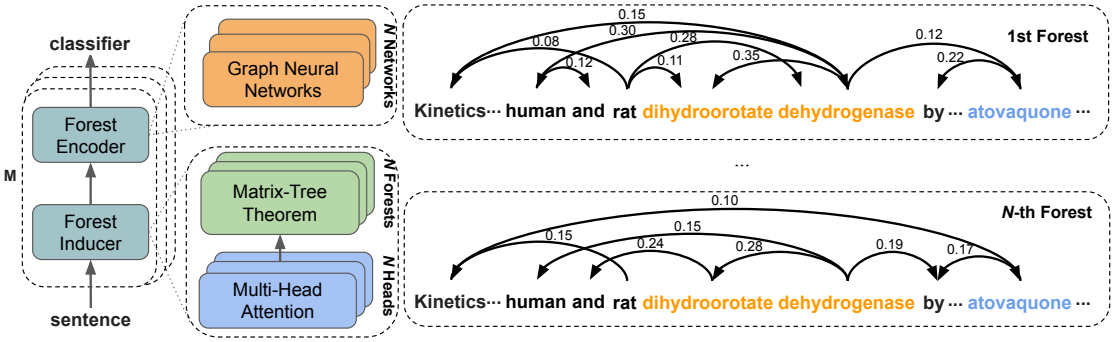


FIGURE 5.8: Overview of our proposed LF-GCN model.

5.7.1 Forest Inducer

Given a sentence $\mathbf{s} = w_1, ..., w_n$, where w_i represents the i-th word, we define a graph $\mathbf{G}$ on n nodes, where each node refers to the word in the $\mathbf{s}$, and the edge (i, j) refers to the dependency between the i-th word (head) to the j-th node (modifier). We denote the contextual output of the sentence $\mathbf{h} \in \mathbb{R}^{n \times d}$ as $\mathbf{h} = \mathbf{h}_1, ..., \mathbf{h}_n$, where $\mathbf{h}_i \in \mathbb{R}^d$ represents the hidden state of the i-th word with a d dimension. We use the bi-directional LSTM to obtain contextual representations of the sentence.

For the graph $\mathbf{G}$, MTT takes the edge scores and root scores as inputs then generates a latent forest by computing the marginal probabilities for each edge. Given the input $\mathbf{h}$ and a weight vector $\boldsymbol{\theta}^k \in \mathbb{R}^m$ of dependencies, where $m \in \mathbb{R}$ represents the number of dependencies for the k-th ($k \in [1, N]$) latent structure, inducing the k-th latent forest for the input $\mathbf{h}$ amounts to finding the latent variables $z_{ij}^k(\mathbf{h}, \boldsymbol{\theta}^k)$ for all edges that satisfy $i \neq j$ and root node whose index equals to 0.

The k-th latent forest induced by MTT contains many non-projective dependency trees, which are denoted by $\mathbf{T}^k$. Let $P(\mathbf{y}|\mathbf{h}; \boldsymbol{\theta}^k)$ denote the conditional probability of a tree $\mathbf{y}$ over $\mathbf{T}^k$.

Following the formulation by Koo et al. (2007), the marginal probability of a dependency edge from i-th word to j-th word for the k-th forest can be expressed as :

$$P(z_{ij}^k = 1) = \sum_{\mathbf{y} \in \mathbf{T}^k : (i,j) \in \mathbf{y}} P(\mathbf{y}|\mathbf{h}; \boldsymbol{\theta}^k) \tag{5.21}$$

We derive two steps to obtain the marginal probabilities expressed in Equation 5.21.

Computing Attention Scores Motivated by Vaswani et al. (2017), we calculate the edge scores by the multi-head attention mechanism, which allows the model to jointly attend to information from different representation subspaces. N attention matrices will be fed into the MTT to obtain N latent forests in order to capture different dependencies in different representation subspaces. The attention matrix for the k-th head is calculated by a function of the query $\mathbf{Q}$ with the corresponding key $\mathbf{K}$. Here $\mathbf{Q}$ and $\mathbf{K}$ are both equal to the contextual representation $\mathbf{h}$. We project $\mathbf{Q}$ and $\mathbf{K}$ to different representation subspaces in order to generate N attention matrices for calculating N latent forests. Formally, the k-th forest $\mathbf{S}^k$ is given by:

$$\mathbf{S}^k = \text{softmax}(\frac{\mathbf{Q}\mathbf{W}^Q \times (\mathbf{K}\mathbf{W}^K)^T}{\sqrt{d}}) \tag{5.22}$$

where $\mathbf{W}^Q \in \mathbb{R}^{d \times d}$ and $\mathbf{W}^K \in \mathbb{R}^{d \times d}$ are parameters for projections. $\mathbf{S}_{ij}^k$ denotes the normalized attention score between the i-th token and the j-th token with $\mathbf{h}_i$ and $\mathbf{h}_j$. Then, we compute the root score $\mathbf{r}_i^k$, which represents the normalized probability of the i-th node to be selected as the root node of the k-th forest:

$$\mathbf{r}_i^k = \mathbf{W}_r^k \mathbf{h}_i \tag{5.23}$$

where $\mathbf{W}_r^k \in \mathbb{R}^{1 \times d}$ is the weight for the projection.

Imposing Structural Constraint Following Koo et al. (2007) and Smith and Smith (2007), we calculate the marginal probability of each dependency edge of the k-th latent forest, by injecting a structural bias on $\mathbf{S}^k$. We assign non-negative weights $\mathbf{P}^k \in \mathbb{R}^{n \times n}$ to the edges as:

$$\mathbf{P}_{ij}^k = \begin{cases} 0 & \text{if } i = j \\ \exp\left(\mathbf{S}_{ij}^k\right) & \text{otherwise} \end{cases} \tag{5.24}$$

where $\mathbf{P}_{ij}^k$ is the weight of the edge between the i-th and the j-th node. We define a Laplacian matrix $\mathbf{L}^k \in \mathbb{R}^{n \times n}$ of $\mathbf{G}$ in Equation 5.25, and its variant $\hat{\mathbf{L}}^k \in \mathbb{R}^{n \times n}$ in Equation 5.26.

$$\mathbf{L}_{ij}^k = \begin{cases} \sum_{i'=1}^{n} \mathbf{P}_{i'j}^k & \text{if } i = j \\ -\mathbf{P}_{ij}^k & \text{otherwise} \end{cases} \tag{5.25}$$

$$\hat{\mathbf{L}}_{ij}^k = \begin{cases} \exp(\mathbf{r}_i^k) & \text{if } i = 1 \\ \mathbf{L}_{ij}^k & \text{if } i > 1 \end{cases} \tag{5.26}$$

We use $\mathbf{A}_{ij}^k$ to denote the marginal probability of the dependency edge between the i-th node and the j-th node. Then, $\mathbf{A}_{ij}^k$ can be derived based on:

$$\begin{aligned}\mathbf{A}^k(z_{ij}=1) = (1-\delta_{1,j})\mathbf{P}_{ij}^k[(\hat{\mathbf{L}}^k)^{-1}]_{ij} \\ -(1-\delta_{i,1})\mathbf{P}_{ij}^k[(\hat{\mathbf{L}}^k)^{-1}]_{ji}\end{aligned} \tag{5.27}$$

where δ is Kronecker delta and $\mathbf{A}^k \in \mathbb{R}^{n\times n}$ can be interpreted as a weighted adjacency matrix for the k-th forest. Now we can feed $\mathbf{A} \in \mathbb{R}^{N\times n\times n}$ into the forest encoder to update the representations of nodes in the latent structure.

5.7.2 Adaptive Pruning Strategy

Prior dependency-based model (Zhang, Qi, and Manning, 2018) also proposes a rule-based method to prune a dependency tree to further improve relation classification performance. However, the weighted adjacency matrix $\mathbf{A}$ is derived based on a continuous relaxation, and such induced structures are not discrete, so the existing rule-based pruning methods are not applicable. Instead, we use α-entmax (Blondel, Martins, and Niculae, 2018; Correia, Niculae, and Martins, 2019) to remove irrelevant information by imposing the sparsity constraints on the adjacency matrix. α-entmax is able to assign exactly zero weights. Therefore, an unnecessary path in the induced latent forests will not be considered by the latent forest encoder. The expression of our soft pruning strategy is described as:

$$\mathbf{A}^k = \alpha\text{-entmax}(\mathbf{A}^k) \tag{5.28}$$

where α is a parameter to control the sparsity of each adjacency matrix. When α=2, the entmax recovers the sparsemax mapping (Martins and Astudillo, 2016). When α=1, it recovers the softmax mapping. Correia, Niculae, and Martins (2019) propose an exact algorithm to learn α automatically. Here we apply k α-entmax to k latent forests, which enables the model to develop different pruning strategies for different latent forests.

5.7.3 Forest Encoder

Given N latent forests generated by the forest inducer, we encode them by using densely-connected graph convolutional networks (Kipf and Welling, 2017; Guo et al., 2019). Formally, given the k-th latent forest, which is represented by the adjacency matrix $\mathbf{A}^k$, the convolution computation for the i-th node at the l-th layer, which takes the representation $\mathbf{h}_i^{l-1}$ from previous layer as input and outputs the updated representations $\mathbf{h}_i^l$, can be defined as:

$$\mathbf{h}_{k_i}^l = \sigma(\sum_{j=1}^{n} \mathbf{A}_{ij}^k \mathbf{W}_k^l \mathbf{h}_i^{l-1} + \mathbf{b}_k^l) \tag{5.29}$$

where $\mathbf{W}_k^l$ and $\mathbf{b}_k^l$ are the weight matrix and bias vector for the k-th latent forest in the l-th layer, respectively. σ is an activation function. $\mathbf{h}_i^0 \in \mathbb{R}^d$ is the initial contextual representation of the i-th node. Then a linear combination layer is used to integrate representations of the N latent forests:

$$\mathbf{h}_{comb} = \mathbf{W}_{comb}\mathbf{h}_{out} + \mathbf{b}_{comb} \tag{5.30}$$

where $\mathbf{h}_{out}$ is the output by concatenating outputs from N separated convolutional layers, *i.e.,* $\mathbf{h}_{out} = [\mathbf{h}^{(1)}; ...; \mathbf{h}^{(N)}] \in \mathbb{R}^{d \times N}$. $\mathbf{W}_{comb} \in \mathbb{R}^{(d \times N) \times d}$ is a weight matrix and $\mathbf{b}_{comb}$ is a bias vector for the linear transformation.

5.8 Experiments of Latent Forests

We conduct experiments on four medical datasets as well as an additional dataset from the news domain to validate the effectiveness of our proposed approach.

5.8.1 Data

We evaluate our LF-GCN model with four datasets on two tasks, namely cross-sentence n-ary relation extraction and sentence-level relation extraction.

For cross-sentence n-ary relation extraction, we use two datasets generated by Peng et al. (2017a), which has 6,987 ternary relation instances and 6,087 binary relation instances extracted from PubMed. The relation label contains five categories, *e.g.,* "sensitivity", "resistance" and "none". Following Linfeng et al. (2018), we define two sub-tasks for a more detailed evaluation, *i.e.,* binary-class n-ary relation extraction and multi-class n-ary relation extraction. For binary-class extraction, we cluster the four relation classes as "Yes" and treat the label "None" as "No".

For sentence-level relation extraction, we follow the experimental settings by Lifeng et al. (2020) on BioCreative Vi CPR (CPR; Krallinger, Rabal, Akhondi, et al. 2017) and Phenotype-Gene relation (PGR; Sousa, Lamúrias, and Couto 2019). The CPR dataset contains the relations between chemical components and human proteins. It has 16,107 training, 10,030 development and 14,269 testing instances, with five regular relations, such as "CPR:3", "CPR:9" and "None" relation. PGR introduces the relations between human phenotypes with human genes, and it contains 11,780 training instances and 219 test instances, with binary class "Yes" and "No" on relation labels.

We also use SemEval-2010 Task 8 (Hendrickx et al., 2010) dataset from the news domain to evaluate the generalization capability of our model. It has 10,717 instances with 9 types of relations, such as "Cause-Effect", "Content-Container", and a special "Other" relation.

5.8.2 Experimental Setup

We tune the hyper-parameters according to the results on the development sets. For the cross-sentence n-ary relation extraction task, we use the same data splits as Linfeng et al. (2018), stochastic gradient descent optimizer with a 0.9 decay rate, and 300-dimensional GloVe. The hidden size of both BiLSTM and GCNs are set as 300.

For cross-sentence task, we report the test accuracy averaged over five cross validation folds (Linfeng et al., 2018) for the cross-sentence n-ary task. For the sentence-level task, we report the $F1$ scores (Lifeng et al., 2020).

5.8.3 Main Results

Results on Cross-Sentence n-ary Task To verify the effectiveness of the model in predicting inter-sentential relations, we compare LF-GCN against state-of-the-art systems on the cross-sentence n-ary relation extraction task (Peng et al., 2017a), as shown in Table 5.11. Previous systems using the same syntax type are grouped together.

Syntax Type	Model	Binary-class				Multi-class	
		Ternary		Binary		Ternary	Binary
		Single	Cross	Single	Cross	Cross	Cross
Full Tree	DAG LSTM (Peng et al., 2017a)	77.9	80.7	74.3	76.5	-	-
	GRN (Linfeng et al., 2018)	80.3	83.2	83.5	83.6	71.7	71.7
	GCN (Zhang, Qi, and Manning, 2018)	84.3	84.8	84.2	83.6	77.5	74.3
Pruned Tree	GCN (Zhang, Qi, and Manning, 2018)	85.8	85.8	83.8	83.7	78.1	73.6
Forest	AGGCN (Guo, Zhang, and Lu, 2019)	87.1	87.0	85.2	85.6	79.7	77.4
	LF-GCN (Ours)	**88.0**	**88.4**	**86.7**	**87.1**	**81.5**	**79.3**

TABLE 5.11: Average test accuracies for binary-class and multi-class n-ary relation extraction. "Ternary" and "Binary" denote drug-gene-mutation tuple and drug-mutation pair, respectively. `Single` and `Cross` indicate that the entities of relations reside in single sentence or multiple sentences, respectively.

Full Tree: models use the 1-best dependency graph constructed by connecting roots of dependency trees correspond to the input sentences. DAG LSTM encodes the graph by using graph-structure LSTM, while GRN and GCN encode it using graph recurrent networks and graph convolutional networks, respectively.
Pruned Tree: model with pruned trees as inputs, whose dependency nodes and edges are removed based on rules (Zhang, Qi, and Manning, 2018). GCN is used to encode the resulting structure.
Forest: model constructs multiple fully-connected weighted graphs based on the multi-head attention (Vaswani et al., 2017), where the graph can be viewed as a dependency forest.

Models with pruned trees as inputs tend to achieve higher results than models with full trees. Intuitively, longer sentences in the cross-sentence task correspond to more complex dependency structures. Using an out-of-domain parser may introduce more noise to the model. Removing the irrelevant nodes and edges of the parse tree enables the model to perform better prediction. However, a rule-based pruning strategy (Zhang, Qi, and Manning, 2018) may not yield optimal performance. In contrast, LF-GCN induces the dependency structure automatically, which can be viewed as a soft pruning strategy learned from the data. Compared to GCN models, our model obtains 2.2 and 2.6 points improvement over the best performing model with pruned trees for the ternary relation extraction. For binary relation extraction, our model achieves accuracy of 86.7 and 87.1 under `Single` and `Cross` settings, respectively, which surpasses the state-of-the-art AGGCN model. We believe that our LF-GCN is able to distill relevant information and filter out noises from the representation for better prediction.

Results on Sentence-Level Task To examine LF-GCN on sentence-level task, we compare LF-GCN with state-of-the-art models on two medical datasets, *i.e.,* CPR (Krallinger, Rabal, Akhondi, et al., 2017) and PGR (Sousa, Lamúrias, and Couto, 2019). These systems are grouped in three types based on the syntactic structure used as shown in Table 5.12 and Table 5.13. Results labeled with "$*$" are obtained based on the re-trained models using their released implementations, as we don't have published results for the dataset.
None: models do not use any pre-trained parsers. Random-DDCNN uses a randomly initialized parser (Dozat and Manning, 2017) fine-tuned by the relation prediction loss. Att-GRU stacks a self-attention layer on top of the gated recurrent units and Bran relies on a bi-affine self-attention model to capture the interactions in the sentence. BioBERT is a pre-trained language representation model for biomedical text.

Syntax Type	Model	$F1$
None	Random-DDCNN (Lifeng et al., 2020)	45.4
	Att-GRU (Liu et al., 2017)	49.5
	Bran (Verga, Strubell, and McCallum, 2018)	50.8
Tree	GCN (Zhang, Qi, and Manning, 2018)	52.2*
	Tree-DDCNN (Lifeng et al., 2020)	50.3
	Tree-GRN (Lifeng et al., 2020)	51.4
Forest	Edgewise-GRN (Song et al., 2019a)	53.4
	KBest-GRN (Song et al., 2019a)	52.4
	AGGCN (Guo, Zhang, and Lu, 2019)	56.7*
	ForestFT-DDCNN (Lifeng et al., 2020)	55.7
	LF-GCN (Ours)	**58.9**

TABLE 5.12: Test results on the CPR dataset. Results on AGGCN and GCN are reproduced based on their released implementation.

Syntax Type	Model	$F1$
None	BioBERT (Lee et al., 2019)	67.2
Tree	BO-LSTM (Lamurias et al., 2019)	52.3
	GCN (Zhang, Qi, and Manning, 2018)	81.3*
	Tree-GRN (Lifeng et al., 2020)	78.9
Forest	Edgewise-GRN (Song et al., 2019a)	83.6
	KBest-GRN (Song et al., 2019a)	85.7
	AGGCN (Guo, Zhang, and Lu, 2019)	89.3*
	ForestFT-DDCNN (Lifeng et al., 2020)	89.3
	LF-GCN (Ours)	**91.9**

TABLE 5.13: Test results on the PGR dataset. Results on AGGCN and GCN are reproduced based on their released implementations.

Tree: models use the 1-best dependency tree. Full trees are encoded by GCN, GRN and DD-CNN, respectively. BO-LSTM only encodes words on the shortest dependency path.
Forest: models leverage the dependency forest. Edgewise-GRN constructs a dependency forest by keeping all the edges with scores greater than a pre-defined threshold. KBest-GRN generates a forest by merging K-bests trees. ForestFT-DDCNN builds a forest by a learnable dependency parser. AGGCN computes attention matrices and treats them as the adjacency matrices of forests.

As shown in Table 5.13, models with full dependency trees or forests as inputs are able to significantly outperform all models that only consider the text sequence including BioBERT, which is trained on a very large-scale medical corpus. These results demonstrate that modeling structure in the input sentence is beneficial to the relation extraction task. Models with dependency forests as inputs yield better performance than those use 1-best dependency trees, which confirms our hypothesis that the error propagation, which is caused by the low parsing accuracy of an out-of-domain parser, can be alleviated by constructing weighted graphs (forests). Compared with models which encode fixed dependency forests that are generated at the data preprocessing stage (Edgewise-GRN and KBest-GRN), models with dynamic forests including

Syntax Type	Model	$F1$
Tree	Tree-GRN (Song et al., 2019a)	84.6
	GCN (Zhang, Qi, and Manning, 2018)	84.8
Forest	ForestFT-DDCNN (Lifeng et al., 2020)	85.5
	AGGCN (Guo, Zhang, and Lu, 2019)	85.7
	KBest-GRN (Song et al., 2019a)	85.8
	Edgewise-GRN (Song et al., 2019a)	**86.3**
	LF-GCN (Ours)	85.7

TABLE 5.14: Test results on the SemEval dataset.

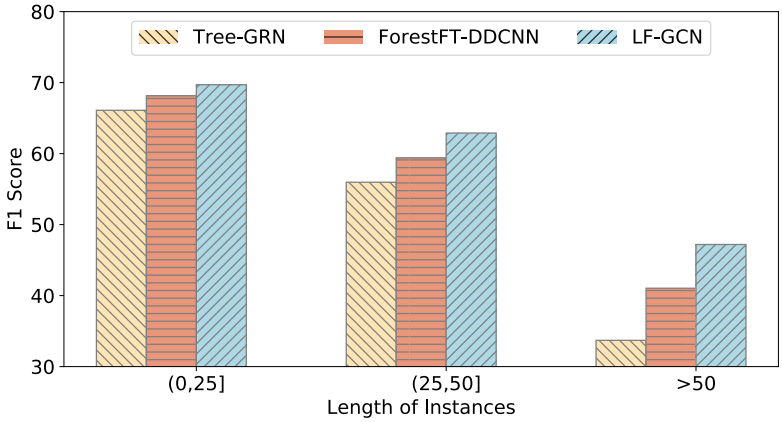


FIGURE 5.9: $F1$ scores against sentence length.

AGGCN, ForestFT-DDCNN and LF-GCN achieve higher performance. On the other hand, LF-GCN also makes predictions without recourse to any pre-trained parsers, while it outperforms Random-DDCNN by a large margin, *i.e.,* 14.5. Furthermore, our LF-GCN model achieves 58.9 and 91.9 scores on CPR and PGR datasets, which are consistently better than all forest generation approaches. These results suggest that the induced latent structure is able to capture task-specific information for better relation extraction.

Results on News Domain LF-GCN can also be used in other domains. Table 5.14 gives the results on SemEval (Hendrickx et al., 2010) dataset from the news domain. Using limited training data, LF-GCN outperforms the models with a dependency tree including Tree-GRN and GCN by almost 1 point and is comparable with the models with dependency forests including AGGCN and ForestFT-DDCNN. This demonstrates that LF-GCN is able to learn a comparable expressive structure compared with the structure generated by an in-domain parser. LF-GCN is 0.6 point worse than Edgewise-GRN. The reason is that the parsing performance for newswire is much more accurate than the biomedical domain. We believe that our model is able to achieve higher performance if more training data is available.

5.8.4 Analysis and Discussion

Performance against Sentence Length To investigate our LF-GCN performance under different sentence lengths, we split the test set of CPR into three categories $((0, 25], (25, 50], > 50)$

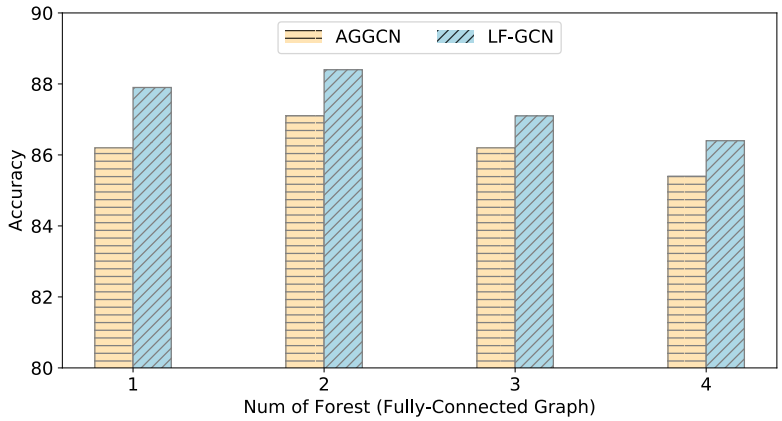


FIGURE 5.10: $F1$ scores against the number of forests under the ⟨ Binary-class, Ternary, `Cross` ⟩ setting shown in Table 5.11. The results on AGGCN are reproduced based on its released implementation.

based on the lengths. As shown in Figure 5.9, we compare our LF-GCN with Tree-GRN (Song et al., 2019a) and ForestFT-DDCNN (Lifeng et al., 2020). In general, LF-GCN outperforms Tree-GRN and ForestFT-DDCNN for each group of instances, showing the effectiveness of our model based on a latent structure induction. The performance gap is enlarged when the instance length increases. Intuitively, longer instances are more challenging since the dependency structure is a more sophisticated tree. These results illustrate that the induced structure is able to capture complex non-local interactions for better prediction.

Performance against Number of Forests Figure 5.10 shows the performance of LF-GCN and AGGCN with different numbers of forests, since AGGCN also leverages the multi-head attention mechanism (Vaswani et al., 2017) to generate multiple weighted graphs. Even though AGGCN is initialized with the 1-best dependency tree generated by a pre-trained parser, LF-GCN consistently outperforms it under the same number of forest without relying on any parsers, where the numbers range from 1 to 4. These results demonstrate that our model is able to construct informative structures only based on the input text.

5.8.5 Case Study

In this section, we use the Chu-Liu-Edmonds (Chu and Liu, 1965) algorithm to extract N non-projective dependency trees from N latent forests, where each forest is expressed by a weighted adjacency matrix in Equation 5.28. Here N equals to 2. We select an instance from the CPR development set, whose relations can be correctly predicted by our LF-GCN. The sentence of this instance is "*VT recurred with the addition of* ***aminophylline****, a competitive* ***adenosine A1-receptor*** *antagonist.*"

Case I: As shown in Figure 5.11, these two dependency trees are able to capture rich interactions between the entities ***Schisandrin B*** (index 0 and 1) and ***DT-diaphorase*** (index 10, 11 and 12), which have a "up regulator" relation, denoted as "CPR:3". For example, the token "enhancing" (index 9), which shares the similar semantic as the gold relation "up regulator", is selected in the path between these two entities. Furthermore, these two trees show different

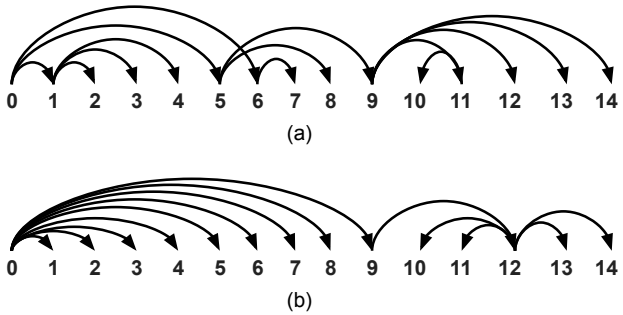


FIGURE 5.11: (a) the first and (b) the second non-projective dependency tree, which are extracted from two latent forests induced by LF-GCN.

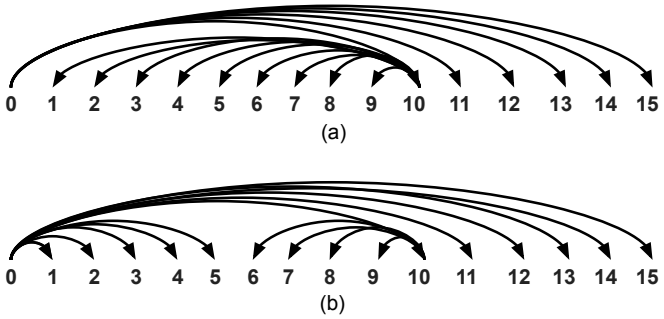


FIGURE 5.12: (a) the first and (b) the second non-projective dependency tree, which are extracted from two latent forests induced by LF-GCN.

dependencies between tokens, which confirms our hypothesis that inducing multiple forests can include more useful information.

Case II: However, as shown in Figure 5.12, many dependency trees induced by structure attention are shallow and do not resemble a linguistic syntax structure. Figure 5.12 shows two shallow trees extracted from the latent forests before imposing sparsity constraints. We observe that the constructed dependency trees tend to pick the first token of the sentence as the root, and all other tokens as the children. Interestingly, even though such trees have little to no structure, the model is still able to predict the correct relation label. We also notice that adding the α-entmax helps to induce deeper and more informative structures. We leave the investigation of this phenomenon as future work.

Chapter 6

Conclusion

In this thesis, we mainly focus on learning with graphs in natural language processing. We explore two directions. The first one is to predict graphs from plain texts and the second one is to integrate graph structures for different downstream tasks, such as natural language generation and relation extraction.

We first investigate how to predict the corresponding semantic graph from the input sentence. Specifically, we focus on the Abstract Meaning Representation (AMR; Banarescu et al. 2013), which is a directed, labelled graph. Such a graph can capture rich semantic level structural information, and are attractive representations useful for semantics-related tasks. AMR parsing is a challenging task as it requires the parser to learn to predict not only concepts, which consist of predicates, lemmas, named entities and co-references, but also a large number of relation types based on relatively sparse training data. We present a novel transition-based system which refines the search space. Experiments show that our parser is able to achieve state-of-the-art performance with a simple architecture and minimal additional resources. We believe our end-to-end system is helpful in practical settings. In the future, we would also like to investigate if it is possible to build a tree to graph transducer, which is able to convert a dependency tree into its corresponding AMR graph.

Then we explore how to leverage semantic graphs or dependency graphs for better natural language generation. The key challenge in graph encoding is how to efficiently learn an effective graph representation. We first introduce the novel densely connected graph convolutional networks (DCGCNs) to learn structural graph representations. Experimental results show that DCGCNs can outperform best reported graph neural networks in the graph-to-sequence learning task. Unlike previous designs of GCNs, DCGCNs scale naturally to significantly more layers without suffering from performance degradation and optimization difficulties, thanks to the introduced dense connectivity mechanism. Such a deep architecture allows the encoder to better capture the rich structural information of a graph, especially when it is large. We further propose the lightweight dynamic graph convolutional networks (LDGCNs) by introducing the novel dynamic fusion mechanism. This mechanism is able to integrate high order information in the graph convolutional operations, allowing rich local and non-local information to be captured. Two novel weight sharing strategies based on the group graph convolutions and weight tied convolutions are further developed to reduce memory usage and model complexity. Compared with existing graph convolutional networks and structured self-attention networks, our model maintains a better balance between parameter efficiency and model capacity by consistently outperforming state-of-the-art models with significantly fewer parameters.

There are multiple venues for future work. One natural question we would like to ask is how to make use of the proposed framework to perform improved graph representation learning for various graph related tasks (Kipf and Welling, 2017). On the other hand, we would also like

to investigate how other NLP applications such as name entity recognition and semantic role labeling can potentially benefit from our proposed approach.

Apart from language generation, relation extraction models also benefit from incorporating structural information. For example, dependency graph are able to capture non-local syntactic relations that are obscure from the surface form alone for better extraction. A key challenge in graph-based models is the error propagation. The dependency structure predicted by an external parser may introduce noises into the model. Therefore, We introduce the novel Attention Guided Graph Convolutional Networks (ACGCNs). Unlike previous approaches, AGGCNs operate directly on the full tree and learn to distill the useful information from it in an end-to-end fashion. For document-level and bio-medical relation extraction tasks, parsing accuracies can drop significantly. This can lead to severe error propagation in downstream relation extraction tasks, offsetting much of the benefit that relation extraction models can obtain by integrating the dependency graphs.

Thus, we introduce a novel latent structure refinement model for document-level relation extraction. Unlike previous approaches that rely on syntactic trees, co-references or heuristics, our proposed model dynamically learns a document-level graph and makes predictions in an end-to-end fashion. For bio-medical relation extraction, we propose a novel model that is able to automatically induce a latent structure, without recourse to any tree supervision or pre-training. Extensive results show that our approach is able to better alleviate the error propagation caused by an out-of-domain dependency parser, giving significantly better results than previous state-of-the-art systems.

Bibliography

Abu-El-Haija, Sami et al. (2018). "N-GCN: Multi-scale graph convolution for semi-supervised node classification". In: *Proc. of UAI*.

Abu-El-Haija, Sami et al. (2019). "MixHop: Higher-Order Graph Convolutional Architectures via Sparsified Neighborhood Mixing". In: *Proc. of ICML*.

Alberti, Chris et al. (2017). "SyntaxNet Models for the CoNLL 2017 Shared Task". In: *arXiv preprint*.

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2015). "Neural Machine Translation by Jointly Learning to Align and Translate". In: *Proc. of ICLR*.

Bai, Shaojie, Zico Kolter, and Vladlen Koltun (2019a). "Deep Equilibrium Models". In: *Proc. of NeurIPS*.

— (2019b). "Trellis Networks for Sequence Modeling". In: *Proc. of ICLR*.

Ballesteros, Miguel and Yaser Al-Onaizan (2017). "AMR Parsing using Stack-LSTMs". In: *Proc. of EMNLP*.

Banarescu, Laura et al. (2013). "Abstract meaning representation for SemBanking". In: *Proc. of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*.

Barzdins, Guntis and Didzis Gosko (2016). "Riga at SemEval-2016 task 8: Impact of Smatch extensions and character-level neural translation on AMR parsing accuracy". In: *Proc. of NAACL-HLT*.

Bastings, Joost et al. (2017). "Graph Convolutional Encoders for Syntax-aware Neural Machine Translation". In: *Proc. of EMNLP*.

Battaglia, P. et al. (2018). "Relational inductive biases, deep learning, and graph networks". In: *ArXiv* abs/1806.01261.

Beck, Daniel, Gholamreza Haffari, and Trevor Cohn (2018). "Graph-to-Sequence Learning using Gated Graph Neural Networks". In: *Proc. of ACL*.

Blondel, Mathieu, André F. T. Martins, and Vlad Niculae (2018). "Learning Classifiers with Fenchel-Young Losses: Generalized Entropies, Margins, and Algorithms". In: *AISTATS*.

Brown, Peter F et al. (1993). "The mathematics of statistical machine translation: Parameter estimation". In: *Computational linguistics* 19.2, pp. 263–311.

Bruna, Joan (2014). "Spectral Networks and Deep Locally Connected Networks on Graphs". In: *Proc. of ICLR*.

Bunescu, Razvan C. and Raymond J. Mooney (2005). "A Shortest Path Dependency Kernel for Relation Extraction". In: *Proc. of EMNLP*.

Buys, Jan and Phil Blunsom (2017). "Robust Incremental Neural Semantic Graph Parsing". In: *Proc. of ACL*.

Cai, Deng and Wai Lam (2019). "Core Semantic First: A Top-down Approach for AMR Parsing". In: *Proc. of EMNLP*.

— (2020a). "AMR Parsing via Graph-Sequence Iterative Inference". In: *Proc. of ACL*.

— (2020b). "Graph Transformer for Graph-to-Sequence Learning". In: *Proc. of AAAI*.

Cai, Rui, Xiaodong Zhang, and Houfeng Wang (2016). "Bidirectional Recurrent Convolutional Neural Network for Relation Classification". In: *Proc. of ACL*.

Cai, Shu and Kevin Knight (2013). "Smatch: an evaluation metric for semantic feature structures". In: *Proc. of ACL*.

Cao, Kris and Stephen Clark (2019). "Factorising AMR generation through syntax". In: *Proc. of NAACL-HLT*.

Chen, Danqi and Christopher Manning (2014). "A fast and accurate dependency parser using neural networks". In: *Proc. of EMNLP*.

Chen, Tianqi et al. (2015). "MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems". In: *arXiv preprint*.

Cheng, Jianpeng, Li Dong, and Mirella Lapata (2016). "Long Short-Term Memory-Networks for Machine Reading". In: *Proc. of EMNLP*.

Cheng, Jianpeng et al. (2017). "Learning structured natural language representations for semantic parsing". In: *Proc. of ACL*.

Cho, Kyunghyun et al. (2014). "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *Proc. of EMNLP*.

Christopoulou, Fenia, Makoto Miwa, and Sophia Ananiadou (2018). "A walk-based model on entity graphs for relation extraction". In: *Proc. of ACL*.

— (2019). "Connecting the Dots: Document-level Neural Relation Extraction with Edge-oriented Graphs". In: *Proc. of EMNLP*.

Chu, Yoeng-Jin and Tseng-Hong Liu (1965). "On the shortest arborescence of a directed graph". In: *Scientia Sinica*.

Correia, Gonçalo M, Vlad Niculae, and André FT Martins (2019). "Adaptively Sparse Transformers". In: *EMNLP*.

Damonte, Marco and Shay B. Cohen (2019). "Structural Neural Encoders for AMR-to-text Generation". In: *Proc. of NAACL-HLT*.

Damonte, Marco, Shay B. Cohen, and Giorgio Satta (2016). "An incremental parser for abstract meaning representation". In: *Proc. of EACL*.

Dauphin, Yann et al. (2016). "Language Modeling with Gated Convolutional Networks". In: *Proc. of ICML*.

De Cao, Nicola, Wilker Aziz, and Ivan Titov (2019). "Question answering by reasoning across documents with graph convolutional networks". In: *Proc. of NAACL-HLT*.

Defferrard, Michaël, Xavier Bresson, and Pierre Vandergheynst (2016). "Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering". In: *Proc. of NeurIPS*.

Dehghani, Mostafa et al. (2019). "Universal Transformers". In: *Proc. of ICLR*.

Denkowski, Michael J. and Alon Lavie (2014). "Meteor Universal: Language Specific Translation Evaluation for Any Target Language". In: *Proc. of WMT@ACL*.

Devlin, Jacob et al. (2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *NAACL-HLT*.

Doddington, George et al. (2004). "The Automatic Content Extraction (ACE) Program - Tasks, Data, and Evaluation". In: *Proc. of LREC*.

Dozat, Timothy and Christopher D Manning (2017). "Deep biaffine attention for neural dependency parsing". In: *ICLR*.

Dyer, Chris et al. (2015). "Transition-based dependency parsing with stack long short-term memory". In: *Proc. of ACL*.

Eisner, Jason (1996). "Three New Probabilistic Models for Dependency Parsing: An Exploration". In: *COLING*.

Elman, Jeffrey (1990). "Finding structure in time". In: *Cognitive science* 14.2, pp. 179–211.

Felix, Hieber et al. (2017). "Sockeye: A toolkit for neural machine translation". In: *arXiv preprint*.

Ferracane, Elisa et al. (2019). "Evaluating Discourse in Structured Text Representations". In: *Proc. of ACL*.

Flanigan, Jeffrey et al. (2014). "A discriminative graph-based parser for the abstract meaning representation". In: *Proc. of ACL*.

Flanigan, Jeffrey et al. (2016a). "CMU at SemEval-2016 task 8: Graph-based AMR parsing with infinite ramp loss". In: *Proc. of SemEval*.

Flanigan, Jeffrey et al. (2016b). "Generation from Abstract Meaning Representation using Tree Transducers". In: *Proc. of NAACL-HLT*.

Foland, William and James H Martin (2017). "Abstract meaning representation parsing using LSTM recurrent neural networks". In: *Proc. of ACL*.

Gehring, Jonas et al. (2017). "Convolutional Sequence to Sequence Learning". In: *Proc. of ICML*.

Girshick, Ross et al. (2014). "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proc. of CVPR*.

Goodman, James, Andreas Vlachos, and Jason Naradowsky (2016). "UCL+Sheffield at SemEval-2016 Task 8: Imitation learning for AMR parsing with an alpha-bound". In: *Proc. of SemEval*.

Gori, Michele, Gabriele Monfardini, and Franco Scarselli (2005). "A new model for learning in graph domains". In: *Proc. of IJCNN*.

Groschwitz, Jonas et al. (2017). "A constrained graph algebra for semantic parsing with AMRs". In: *Proc. of IWCS*.

Groschwitz, Jonas et al. (2018). "AMR Dependency Parsing with a Typed Semantic Algebra". In: *Proc. of ACL*.

Gross, Jonathan, Jay Yellen, and Ping Zhang (2013). *Handbook of Graph Theory, Second Edition*. Chapman & Hall/CRC. ISBN: 1439880182, 9781439880180.

Gu, Jinghang et al. (2017). "Chemical-induced disease relation extraction via convolutional neural network". In: *Database: The Journal of Biological Databases and Curation* 2017.

Guo, Zhijiang and Wei Lu (2018). "Better Transition-Based AMR Parsing with a Refined Search Space". In: *Proc. of EMNLP*.

Guo, Zhijiang, Yan Zhang, and Wei Lu (2019). "Attention Guided Graph Convolutional Networks for Relation Extraction". In: *Proc. of ACL*.

Guo, Zhijiang et al. (2019). "Densely connected graph convolutional networks for graph-to-sequence learning". In: *Transactions of the Association for Computational Linguistics* 7, pp. 297–312.

Guo, Zhijiang et al. (2020). "Learning Latent Forests for Medical Relation Extraction". In: *Proc. of IJCAI*.

Gupta, Pankaj et al. (2018). "Neural Relation Extraction Within and Across Sentence Boundaries". In: *AAAI*.

Hamilton, William L., Rex Ying, and Jure Leskovec (2017). "Inductive Representation Learning on Large Graphs". In: *Proc. of NeurIPS*.

He, Kaiming et al. (2016). "Deep Residual Learning for Image Recognition". In: *Proc. of CVPR*.

Henaff, Mikael, Joan Bruna, and Yann LeCun (2015). "Deep Convolutional Networks on Graph-Structured Data". In: *arXiv preprint*.

Henderson, James et al. (2013). "Multilingual joint parsing of syntactic and semantic dependencies with a latent variable model". In: *Computational linguistics* 39.4, pp. 949–998.

Hendrickx, Iris et al. (2010). "SemEval-2010 Task 8: Multi-Way Classification of Semantic Relations between Pairs of Nominals". In: *SemEval@ACL*.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long short-term memory". In: *Neural computation* 9.8, pp. 1735–1780.

Hovy, E. et al. (2006). "OntoNotes: The 90% Solution". In: *Proc. of NAACL-HLT*.

Howard, Andrew G. et al. (2017). "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". In: *ArXiv* abs/1704.04861.

Huang, Binxuan and K. Carley (2019). "Syntax-Aware Aspect Level Sentiment Classification with Graph Attention Networks". In: *Proc. of EMNLP*.

Huang, Gao et al. (2017). "Densely Connected Convolutional Networks". In: *Proc. of CVPR*.

Jia, Robin, Cliff Wong, and Hoifung Poon (2019). "Document-Level N-ary Relation Extraction with Multiscale Representation Learning". In: *NAACL-HLT*.

Jie, Zhanming and Wei Lu (2019). "Dependency-Guided LSTM-CRF for Named Entity Recognition". In: *Proc. of EMNLP*.

Jurafsky, Dan and J. Martin (2020). "Speech and Language Processing (3rd Edition)". In:

Kim, Yoon et al. (2017). "Structured Attention Networks". In: *ICLR*.

Kingma, Diederik P. and Jimmy Ba (2015). "Adam: A Method for Stochastic Optimization". In: *Proc. of ICLR*.

Kipf, Thomas N. and Max Welling (2017). "Semi-Supervised Classification with Graph Convolutional Networks". In: *Proc. of ICLR*.

Koehn, Philipp (2004). "Statistical Significance Tests for Machine Translation Evaluation". In: *Proc. of EMNLP*.

Koehn, Philipp, Franz Josef Och, and Daniel Marcu (2003). "Statistical Phrase-Based Translation". In: *Proc. of NAACL-HLT*.

Koehn, Philipp et al. (2007). "Moses: Open Source Toolkit for Statistical Machine Translation". In: *Proc. of ACL(Demo)*.

Konstas, Ioannis and Mirella Lapata (2012). "Unsupervised Concept-to-text Generation with Hypergraphs". In: *Proc. of NAACL-HLT*.

— (2013). "Inducing Document Plans for Concept-to-Text Generation". In: *Proc. of EMNLP*.

Konstas, Ioannis et al. (2017). "Neural AMR: Sequence-to-Sequence models for parsing and generation". In: *Proc. of ACL*.

Koo, Terry K et al. (2007). "Structured Prediction Models via the Matrix-Tree Theorem". In: *EMNLP*.

Krallinger, Martin, Obdulia Rabal, Saber A Akhondi, et al. (2017). "Overview of the BioCreative VI chemical-protein interaction Track". In: *BioCreative challenge evaluation workshop*.

Lafferty, John, Andrew McCallum, and Fernando C. N. Pereira (2001). "Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data". In: *Proc. of ICML*.

Lample, Guillaume et al. (2016). "Neural architectures for named entity recognition". In: *Proc. of NAACL-HLT*.

Lamurias, Andre et al. (2019). "BO-LSTM: classifying relations via long short-term memory networks along biomedical ontologies". In: *BMC bioinformatics*.

Lan, Zhen-Zhong et al. (2020). "ALBERT: A Lite BERT for Self-supervised Learning of Language Representations". In: *Proc. of ICLR*.

LeCun, Yann and Yoshua Bengio (1995). "Convolutional Networks for Images Speech and Time Series". In:

Lee, Chen-Yu et al. (2015). "Deeply-Supervised Nets". In: *Proc. of AISTATS*.

Lee, Jinhyuk et al. (2019). "BioBERT: a pre-trained biomedical language representation model for biomedical text mining". In: *Bioinformatics*.

Lee, Kenton et al. (2017). "End-to-end Neural Coreference Resolution". In: *Proc. of EMNLP*.

Li, Jiao et al. (2016a). "BioCreative V CDR task corpus: a resource for chemical disease relation extraction". In: *Database : the journal of biological databases and curation* 2016.

Li, Qimai, Zhichao Han, and Xiao-Ming Wu (2018). "Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning". In: *Proc. of AAAI*.

Li, Xiang et al. (2019). "Selective Kernel Networks". In: *Proc. of CVPR*.

Li, Yujia et al. (2016b). "Gated Graph Sequence Neural Networks". In: *Proc. of ICLR*.

Li, Zhiheng et al. (2016c). "CIDExtractor: A chemical-induced disease relation extraction system for biomedical literature". In: *Proc. of BIBM*.

Liao, Kexin, Logan Lebanoff, and Fei Liu (2018). "Abstract Meaning Representation for Multi-Document Summarization". In: *Proc. of COLING*.

Lifeng, Jin et al. (2020). "Relation Extraction Exploiting Full Dependency Forests". In: *AAAI*.

Lin, Bill Yuchen et al. (2019). "KagNet: Knowledge-Aware Graph Networks for Commonsense Reasoning." In: *Proc. of EMNLP*.

Lindemann, Matthias, Jonas Groschwitz, and Alexander Koller (2019). "Compositional Semantic Parsing Across Graphbanks". In: *Proc. of ACL*.

Linfeng, Song et al. (2018). "N-ary Relation Extraction using Graph State LSTM". In: *Proc. of EMNLP*.

Ling, Wang et al. (2015). "Two/too simple adaptations of Word2Vec for syntax problems". In: *Proc. of NAACL-HLT*.

Liu, Jiangming, Shay B. Cohen, and Mirella Lapata (2018). "Discourse Representation Structure Parsing". In: *Proc. of ACL*.

— (2019). "Discourse Representation Parsing for Sentences and Documents". In: *Proc of. ACL*.

Liu, Sijia et al. (2017). "Attention-based Neural Networks for Chemical Protein Relation Extraction". In: *Proceedings of the BioCreative VI Workshop*.

Liu, Yang and Mirella Lapata (2018). "Learning Structured Text Representations". In: *TACL*.

Liu, Yang, Ivan Titov, and Mirella Lapata (2019). "Single Document Summarization as Tree Induction". In: *Proc. of NAACL-HLT*.

Liu, Yang et al. (2015). "A Dependency-Based Neural Network for Relation Classification". In: *Proc. of ACL*.

Liu, Yijia et al. (2018). "An AMR Aligner Tuned by Transition-based Parser". In: *Proc. of EMNLP*.

Loper, Edward and Steven Bird (2002). "NLTK: The Natural Language Toolkit". In: *CoRR* cs.CL/0205028.

Lu, Wei and Hwee Tou Ng (2011). "A Probabilistic Forest-to-String Model for Language Generation from Typed Lambda Calculus Expressions". In: *Proc. of EMNLP*.

Lu, Wei, Hwee Tou Ng, and Wee Sun Lee (2009). "Natural Language Generation with Tree Conditional Random Fields". In: *Proc. of EMNLP*.

Lu, Wei et al. (2008). "A Generative Model for Parsing Natural Language to Meaning Representations". In: *Proc. of EMNLP*.

Luan, Sitao et al. (2019). "Break the Ceiling: Stronger Multi-scale Deep Graph Convolutional Networks". In: *Proc. of NeurIPS*.

Lyu, Chunchuan and Ivan Titov (2018). "AMR Parsing as Graph Prediction with Latent Alignment". In: *Proc. of ACL*.

Manning, Christopher D and Hinrich Schütze (1999). *Foundations of statistical natural language processing*. MIT press.

Manning, Christopher D. et al. (2014). "The Stanford CoreNLP Natural Language Processing Toolkit". In: *Demo@ACL*.

Marcheggiani, Diego and Laura Perez-Beltrachini (2018). "Deep Graph Convolutional Encoders for Structured Data to Text Generation". In: *Proceedings of the 11th International Conference on Natural Language Generation*.

Marcheggiani, Diego and Ivan Titov (2017). "Encoding Sentences with Graph Convolutional Networks for Semantic Role Labeling". In: *Proc. of EMNLP*.

Martins, André F. T. and Ramón Fernández Astudillo (2016). "From Softmax to Sparsemax: A Sparse Model of Attention and Multi-Label Classification". In: *ICML*.

McDonald, Ryan T. et al. (2005). "Simple Algorithms for Complex Relation Extraction with Applications to Biomedical IE". In: *Proc. of ACL*.

Mintz, Mike et al. (2009). "Distant supervision for relation extraction without labeled data". In: *Proc. of ACL*.

Misra, Dipendra Kumar and Yoav Artzi (2016). "Neural shift-reduce CCG semantic parsing". In: *Proc. of EMNLP*.

Mitra, A. and Chitta Baral (2016). "Addressing a Question Answering Challenge by Combining Statistical Methods with Inductive Rule Learning and Reasoning". In: *Proc. of AAAI*.

Miwa, Makoto and Mohit Bansal (2016). "End-to-End Relation Extraction using LSTMs on Sequences and Tree Structures". In: *Proc. of ACL*.

Morris, Christopher et al. (2019). "Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks". In: *Proc. of AAAI*.

Nair, Vinod and Geoffrey E. Hinton (2010). "Rectified linear units improve restricted boltzmann machines". In: *Proc. of ICML*.

Nan, Guoshun et al. (2020). "Reasoning with Latent Structure Refinement for Document-Level Relation Extraction". In: *Proc. of ACL*.

Napoles, Courtney, Matthew R. Gormley, and Benjamin Van Durme (2012). "Annotated Gigaword". In: *AKBC-WEKEX@NAACL-HLT*.

Naseem, Tahira et al. (2019). "Rewarding Smatch: Transition-Based AMR Parsing with Reinforcement Learning". In: *Proc. of ACL*.

Nguyen, Dat Quoc and Karin M. Verspoor (2018). "Convolutional neural networks for chemical-disease relation extraction are improved with character-based word embeddings". In: *Proc. of BioNLP*.

Nguyen, Thien Huu and Ralph Grishman (2015). "Relation Extraction: Perspective from Convolutional Neural Networks". In: *Proc. of VS@NAACL-HLT*.

Nivre, Joakim (2003). "An efficient algorithm for projective dependency parsing". In: *Proc. of IWPT*.

— (2008). "Algorithms for Deterministic Incremental Dependency Parsing". In: *Computational Linguistics* 34, pp. 513–553.

— (2009). "Non-projective dependency parsing in expected linear time". In: *Proc. of AFNLP*.

Nivre, Joakim and Ryan McDonald (2008). "Integrating graph-based and transition-based dependency parsers". In: *Proc. of ACL*.

Noord, Rik van and Johan Bos (2017). "Neural semantic parsing by character-based translation: Experiments with abstract meaning representations". In: *arXiv preprint arXiv:1705.09980*.

Panyam, Nagesh C et al. (2018). "Exploiting graph kernels for high performance biomedical relation extraction". In: *Journal of Biomedical Semantics* 9.1, p. 7.

Papineni, Kishore et al. (2002). "Bleu: a Method for Automatic Evaluation of Machine Translation". In: *Proc. of ACL*.

Peng, Nanyun et al. (2017a). "Cross-Sentence N-ary Relation Extraction with Graph LSTMs". In: *Transactions of the Association for Computational Linguistics* 5, pp. 101–115.

Peng, Xiaochang and Daniel Gildea (2016). "UofR at SemEval-2016 Task 8: Learning Synchronous Hyperedge Replacement Grammar for AMR Parsing". In: *Proc. of SemEval@NAACL-HLT*.

Peng, Xiaochang, Daniel Gildea, and Giorgio Satta (2018). "AMR Parsing with Cache Transition Systems". In: *Proc. of AAAI*.

Peng, Xiaochang, Linfeng Song, and Daniel Gildea (2015). "A synchronous hyperedge replacement grammar based approach for AMR parsing". In: *Proc. of CoNLL*.

Peng, Xiaochang et al. (2017b). "Addressing the data sparsity issue in neural AMR parsing". In: *Proc. of EACL*.

Peng, Xiaochang et al. (2018). "Sequence-to-sequence Models for Cache Transition Systems". In: *Proc. of ACL*.

Peng, Yifan, Chih-Hsuan Wei, and Zhiyong Lu (2016). "Improving chemical disease relation extraction with rich features and weakly labeled data". In: *Journal of Cheminformatics* 8.

Pennington, Jeffrey, Richard Socher, and Christopher D. Manning (2014). "Glove: Global Vectors for Word Representation". In: *Proc. of EMNLP*.

Peters, Matthew E. et al. (2018). "Deep contextualized word representations". In: *Proc. of NAACL-HLT*.

Popovic, Maja (2017). "chrF++: words helping character n-grams". In: *Proc. of WMT@ACL*.

Pourdamghani, Nima, Kevin Knight, and Ulf Hermjakob (2016). "Generating English from Abstract Meaning Representations". In: *Proc. of INLG*.

Pourdamghani, Nima et al. (2014). "Aligning English strings with abstract meaning representation graphs". In: *Proc. of EMNLP*.

Pust, Michael et al. (2015). "Parsing English into abstract meaning representation using syntax-based machine translation". In: *Proc. of EMNLP*.

Quirk, Chris and Hoifung Poon (2017). "Distant Supervision for Relation Extraction beyond the Sentence Boundary". In: *Proc. of EACL*.

Rennie, S. et al. (2017). "Self-Critical Sequence Training for Image Captioning". In: *Proc. of CVPR*.

Ribeiro, Leonardo Filipe Rodrigues, Claire Gardent, and Iryna Gurevych (2019). "Enhancing AMR-to-Text Generation with Dual Graph Representations". In: *Proc. of EMNLP*.

Riedel, Sebastian, Limin Yao, and Andrew McCallum (2010). "Modeling Relations and Their Mentions without Labeled Text". In: *Proc. of ECML/PKDD*.

Rink, Bryan and Sanda M. Harabagiu (2010). "UTD: Classifying Semantic Relations by Combining Lexical and Semantic Resources". In: *SemEval@ACL*.

Ross, S., G. Gordon, and J. Bagnell (2011). "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning". In: *Proc. of AISTATS*.

Sahu, Sunil Kumar et al. (2019). "Inter-sentence Relation Extraction with Document-level Graph Convolutional Neural Network". In: *Proc. of ACL*.

Santoro, Adam et al. (2017). "A simple neural network module for relational reasoning". In: *Proc. of NeurIPS*.

Scarselli, Franco et al. (2009). "The Graph Neural Network Model". In: *IEEE Transactions on Neural Networks* 20.1.

Schuster, Mike and Kuldip K. Paliwal (1997). "Bidirectional recurrent neural networks". In: *IEEE Trans. Signal Processing* 45, pp. 2673–2681.

See, A., P. Liu, and Christopher D. Manning (2017). "Get To The Point: Summarization with Pointer-Generator Networks". In: *Proc. of ACL*.

Sennrich, Rico, Barry Haddow, and Alexandra Birch (2016). "Neural Machine Translation of Rare Words with Subword Units". In: *Proc. of ACL*.

Shi, Tianze, Liang Huang, and Lillian Lee (2017). "Fast(er) Exact Decoding and Global Training for Transition-Based Dependency Parsing via a Minimal Feature Set". In: *Proc. of EMNLP*.

Smith, David A. and Noah A. Smith (2007). "Probabilistic Models of Nonprojective Dependency Trees". In: *EMNLP*.

Socher, Richard, John Bauer, Christopher D Manning, et al. (2013). "Parsing with compositional vector grammars". In: *Proc. of ACL*.

Song, Linfeng et al. (2016). "AMR-to-text generation as a Traveling Salesman Problem". In: *Proc. of EMNLP*.

Song, Linfeng et al. (2017). "AMR-to-text Generation with Synchronous Node Replacement Grammar". In: *Proc. of ACL*.

Song, Linfeng et al. (2018a). "A Graph-to-Sequence Model for AMR-to-Text Generation". In: *Proc. of ACL*.

Song, Linfeng et al. (2018b). "Exploring graph-structured passage representation for multi-hop reading comprehension with graph neural networks". In: *arXiv preprint arXiv:1809.02040*.

Song, Linfeng et al. (2019a). "Leveraging Dependency Forest for Neural Medical Relation Extraction". In: *Proc. of EMNLP-IJCNLP*.

Song, Linfeng et al. (2019b). "Semantic Neural Machine Translation Using AMR". In: *Transactions of the Association for Computational Linguistics* 7, pp. 19–31.

Sorokin, Daniil and Iryna Gurevych (2017). "Context-aware representations for knowledge base relation extraction". In: *Proc. of EMNLP*.

Sousa, Diana, André Lamúrias, and Francisco M Couto (2019). "A Silver Standard Corpus of Human Phenotype-Gene Relations". In: *Proc. of NAACL-HLT*.

Surdeanu, Mihai et al. (2012). "Multi-instance Multi-label Learning for Relation Extraction". In: *Proc. of EMNLP-CoNLL*.

Sutskever, Ilya, Oriol Vinyals, and Quoc V Le (2014). "Sequence to sequence learning with neural networks". In: *Proc. of NeurIPS*.

Swayamdipta, Swabha et al. (2016). "Greedy, joint syntactic-semantic parsing with Stack LSTMs". In: *Proc. of SemEval*.

Szubert, Ida et al. (2020). "The Role of Reentrancies in Abstract Meaning Representation Parsing". In: *Proc. of Findings of EMNLP*.

Tai, Kai Sheng, Richard Socher, and Christopher D. Manning (2015). "Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks". In: *Proc. of ACL*.

Tu, Ming et al. (2019). "Multi-hop Reading Comprehension across Multiple Documents by Reasoning over Heterogeneous Graphs". In: *Proc. of ACL*.

Tu, Zhaopeng et al. (2016). "Modeling Coverage for Neural Machine Translation". In: *Proc. of ACL*.

Tutte, William Thomas (1984). *Graph theory*.

Vaswani, Ashish et al. (2017). "Attention Is All You Need". In: *Proc. of NeurIPS*.

Veličković, Petar et al. (2018). "Graph Attention Networks". In: *Proc. of ICLR*.

Verga, Patrick, Emma Strubell, and Andrew McCallum (2018). "Simultaneously Self-Attending to All Mentions for Full-Abstract Biological Relation Extraction". In: *Proc. of NAACL-HLT*.

Vilares, David and Carlos Gómez-Rodríguez (2018). "A Transition-based Algorithm for Unrestricted AMR Parsing". In: *Proc. of NAACL-HLT*.

Vogel, Stephan, Hermann Ney, and Christoph Tillmann (1996). "HMM-based word alignment in statistical translation". In: *Proc. of the 16th conference on Computational linguistics-Volume 2*.

Vu, Ngoc Thang et al. (2016). "Combining Recurrent and Convolutional Neural Networks for Relation Classification". In: *Proc. of NAACL-HLT*.

Wang, Chuan, N. Xue, and Sameer Pradhan (2015a). "Boosting Transition-based AMR Parsing with Refined Actions and Auxiliary Analyzers". In: *Proc. of ACL*.

Wang, Chuan and Nianwen Xue (2017). "Getting the Most out of AMR Parsing". In: *Proc. of EMNLP*.

Wang, Chuan, Nianwen Xue, and Sameer Pradhan (2015b). "A transition-based algorithm for AMR parsing". In: *Proc. of NAACL-HLT*.

Wang, Chuan et al. (2016a). "CAMR at SemEval-2016 task 8: An extended transition-based AMR parser". In: *Proc. of SemEval*.

Wang, Hong et al. (2019). "Fine-tune Bert for DocRED with Two-step Process". In: *arXiv preprint arXiv:1909.11898*.

Wang, Linlin et al. (2016b). "Relation Classification via Multi-Level Attention CNNs". In: *Proc. of ACL*.

Wang, Tianming, Xiaojun Wan, and Hanqi Jin (2020). "AMR-To-Text Generation with Graph Transformer". In: *Transactions of the Association for Computational Linguistics* 8, pp. 19–33.

Welbl, Johannes, Pontus Stenetorp, and Sebastian Riedel (2018). "Constructing datasets for multi-hop reading comprehension across documents". In: *Transactions of the Association for Computational Linguistics* 6, pp. 287–302.

Werling, Keenon, Gabor Angeli, and Christopher Manning (2015). "Robust subgraph generation improves abstract meaning representation parsing". In: *Proc. of ACL*.

Wu, Felix et al. (2019a). "Pay less attention with lightweight and dynamic convolutions". In: *Proc. of ICLR*.

Wu, Ye et al. (2019b). "RENET: A Deep Learning Approach for Extracting Gene-Disease Associations from Literature". In: *International Conference on Research in Computational Molecular Biology*. Springer, pp. 272–284.

Xie, Saining et al. (2017). "Aggregated Residual Transformations for Deep Neural Networks". In: *Proc. of CVPR*.

Xu, Kelvin et al. (2015a). "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention". In: *Proc. of ICML*.

Xu, Keyulu et al. (2018). "Representation Learning on Graphs with Jumping Knowledge Networks". In: *Proc. of ICML*.

Xu, Kun et al. (2015b). "Semantic Relation Classification via Convolutional Neural Networks with Simple Negative Sampling". In: *Proc. of EMNLP*.

Xu, Yan et al. (2015c). "Classifying Relations via Long Short Term Memory Networks along Shortest Dependency Paths". In: *Proc. of EMNLP*.

Yamada, Hiroyasu and Yuji Matsumoto (2003). "Statistical dependency analysis with support vector machines". In: *Proc. of IWPT*.

Yang, Hsiu-Wei et al. (2019). "Aligning Cross-Lingual Entities with Multi-Aspect Information". In: *Proc. of EMNLP*.

Yao, Yuan et al. (2019). "DocRED: A Large-Scale Document-Level Relation Extraction Dataset". In: *Proc. of ACL*.

Yu, Mo et al. (2017). "Improved Neural Relation Detection for Knowledge Base Question Answering". In: *Proc. of ACL*.

Zelenko, Dmitry, Chinatsu Aone, and Anthony Richardella (2002). "Kernel Methods for Relation Extraction". In: *Proc. of EMNLP*.

Zeng, Daojian et al. (2014). "Relation Classification via Convolutional Deep Neural Network". In: *Proc. of COLING*.

Zhang, Sheng et al. (2019a). "AMR Parsing as Sequence-to-Graph Transduction". In: *Proc. of ACL*.

Zhang, Sheng et al. (2019b). "Broad-Coverage Semantic Parsing as Transduction". In: *Proc. of EMNLP*.

Zhang, Xiangyu et al. (2017a). "ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices". In: *Proc. of CVPR*.

Zhang, Yan et al. (2020). "Lightweight, Dynamic Graph Convolutional Networks for AMR-to-Text Generation". In: *Proc. of EMNLP*.

Zhang, Yu, Zhenghua Li, and Min Zhang (2020). "Efficient Second-Order TreeCRF for Neural Dependency Parsing". In: *Proc. of ACL*.

Zhang, Yue, Qi Liu, and Linfeng Song (2018). "Sentence-State LSTM for Text Representation". In: *Proc. of ACL*.

Zhang, Yuhao, Peng Qi, and Christopher D. Manning (2018). "Graph Convolution over Pruned Dependency Trees Improves Relation Extraction". In: *Proc. of EMNLP*.

Zhang, Yuhao et al. (2017b). "Position-aware Attention and Supervised Data Improve Slot Filling". In: *Proc. of EMNLP*.

Zheng, Wei et al. (2018). "An effective neural model extracting document level chemical-induced disease relations from biomedical literature". In: *Journal of biomedical informatics* 83, pp. 1–9.

Zhou, Junsheng et al. (2016a). "AMR parsing with an incremental joint model". In: *Proc. of EMNLP*.

Zhou, Peng et al. (2016b). "Attention-Based Bidirectional Long Short-Term Memory Networks for Relation Classification". In: *Proc. of ACL*.

Zhou, Qiji et al. (2020). "AMR Parsing with Latent Structural Information". In: *Proc. of ACL*.

Zhu, Hao et al. (2019). "Graph Neural Networks with Generated Parameters for Relation Extraction". In: *Proc. of ACL*.