Chapter 21
Graph Neural Networks in Natural Language Processing

Bang Liu, Lingfei Wu

Abstract Natural language processing (NLP) and understanding aim to read from unformatted text to accomplish different tasks. While word embeddings learned by deep neural networks are widely used, the underlying linguistic and semantic structures of text pieces cannot be fully exploited in these representations. Graph is a natural way to capture the connections between different text pieces, such as entities, sentences, and documents. To overcome the limits in vector space models, researchers combine deep learning models with graph-structured representations for various tasks in NLP and text mining. Such combinations help to make full use of both the structural information in text and the representation learning ability of deep neural networks. In this chapter, we introduce the various graph representations that are extensively used in NLP, and show how different NLP tasks can be tackled from a graph perspective. We summarize recent research works on graph-based NLP, and discuss two case studies related to graph-based text clustering, matching, and multi-hop machine reading comprehension in detail. Finally, we provide a synthesis about the important open problems of this subfield.

21.1 Introduction

Language serves as a cornerstone of human cognition. Enable machines to understand natural language is at the very heart of machine intelligence. Natural language processing (NLP) concerns with the interaction between machines and human languages. It is a critical subfield of computer science, linguistics, and artificial intelligence (AI). Ever since the early research about machine translation in the 1950s

Bang Liu
Department of Computer Science and Operations Research, University of Montreal, e-mail: bang.liu@umontreal.ca

Lingfei Wu
JD.COM Silicon Valley Research Center, e-mail: lwu@email.wm.edu

until nowadays, NLP has been playing an essential role in the research of machine learning and artificial intelligence.

NLP has a wide range of applications in the life and business of modern society. Critical NLP applications include but not limited to: machine translation applications that aim to translate text or speech from a source language to another target language (e.g., Google Translation, Yandex Translate); chatbots or virtual assistants that conduct an on-line chat conversation with a human agent (e.g., Apple Siri, Microsoft Cortana, Amazon Alexa); search engines for information retrieval (e.g., Google, Baidu, Bing); question answering (QA) and machine reading comprehension in different fields and applications (e.g., open-domain question answering in search engines, medical question answering); knowledge graphs and ontologies that extract and represent knowledge from multi-sources to improve various applications (e.g., DBpedia (Bizer et al, 2009), Google Knowledge Graph); and recommender systems in E-commerce based on text analysis (e.g., E-commerce recommendation in Alibaba and Amazon). Therefore, AI breakthroughs in NLP are big for business.

Two crucial research problems lie at the core of NLP: i) how to represent natural language texts in a format that computers can read; and ii) how to compute based on the input format to understand the input text pieces. We observe that researchers' ideas on representing and modeling text keep evolving during the long history of NLP development.

Up to the 1980s, most NLP systems were symbolic-based. Different text pieces were considered as symbols, and the models for various NLP tasks were implemented based on complex sets of hand-written rules. For example, classic rule-based machine translation (RBMT) involves a host of rules defined by linguists in grammar books. Such systems include Systran, Reverso, Prompt, and LOGOS (Hutchins, 1995). Rule-based approaches with symbolic representations are fast, accurate, and explainable. However, acquiring the rules for different tasks is difficult and needs extensive expert efforts.

Starting in the late 1980s, statistical machine learning algorithms brought revolution to NLP research. In statistical NLP systems, usually a piece of text is considered as a bag of its words, disregarding grammar and even word order but keeping multiplicity (Manning and Schutze, 1999). Many of the notable early successes occurred in machine translation due to statistical models were developed. Statistical systems were able to take advantage of multilingual textual corpora. However, it is hard to model the semantic structure and information of human language by simply considering the text as a bag of words.

Since the early 2010s, the field of NLP has shifted to neural networks and deep learning, where word embeddings techniques such as Word2Vec (Mikolov T, 2013) or GloVe (Pennington et al, 2014) were developed to represent words as fixed vectors. We have also witnessed an increase in end-to-end learning for tasks such as question answering. Besides, by representing text as a sequence of word embedding vectors, different neural network architectures, such as vanilla recurrent neural networks (Pascanu et al, 2013), Long Short-Term Memory (LSTM) networks (Greff et al, 2016), or convolutional neural networks (Dos Santos and Gatti, 2014), were

applied to model text. Deep learning has brought a new revolution in NLP, greatly improving the performance of various tasks.

In 2018, Google introduced a neural network-based technique for NLP pre-training called Bidirectional Encoder Representations from Transformers (BERT) (Devlin et al, 2019). This model has enabled many NLP tasks to achieve superhuman performance in different benchmarks and has spawned a series of follow-up studies on pre-training large-scale language models (Qiu et al, 2020b). In such approaches, the representations of words are contextual sensitive vectors. By taking the contextual information into account, we can model the polysemy of words. However, large-scale pre-trained language models require massive consumption of data and computing resources. Besides, existing neural network-based models lack explainability or transparency, which can be a major drawback in health, education, and finance domains.

Along with the evolving history of text representations and computational models, from symbolic representations to contextual-sensitive embeddings, we can see an increase of semantical and structural information in text modeling. A key question is: how to further improve the representation of various text pieces and the computational models for different NLP tasks? We argue that representing text as graphs and applying graph neural networks to NLP applications is a highly promising research direction. Graphs are of great significance to NLP research. The reasons are multi-aspect, which will be illustrated in the following.

First, our world consists of things and the relations between them. The ability to draw logical conclusions about how different things are related to one another, or so-called relational reasoning, is central to both human and machine intelligence. In NLP, understanding human language also requires modeling different text pieces and reasoning over their relations. Graph provides a unified format to represent things and the relations between them. By modeling text as graphs, we can characterize the syntactic and semantic structures of different texts and perform explainable reasoning and inference over such representations.

Second, the structure of languages is intrinsically compositional, hierarchical, and flexible. From corpus to documents, paraphrases, sentences, phrases, and words, different text pieces form a hierarchical semantic structure, in which a higher-level semantic unit (e.g., a sentence) can be further decomposed into more fine-grained units (e.g., phrases and words). Such structural nature of human languages can be characterized by tree structures. Furthermore, due to the flexibility of languages, the same meaning can be expressed in different sentences, such as active and passive voices. However, we can unify the representation of varying sentences by semantic graphs like Abstract Meaning Representation (AMR) (Schneider et al, 2015) to make NLP models more robust.

Last but not least, graphs have always been extensively utilized and formed an essential part of NLP applications ranging from syntax-based machine translation, knowledge graph-based question answering, abstract meaning representation for common sense reasoning tasks, and so on. On the other hand, with the vigorous research on graph neural networks, the recent research trend of combining graph neural networks and NLP has become more and more prosperous. Moreover, by uti-

lizing the general representation ability of graphs, we can incorporate multi-modal information (e.g., images or videos) to NLP, integrating different signals, modeling the world contexts and dynamics, and jointly learning multi-tasks.

In this chapter, we present a brief overview of the status of graphs in NLP. We will introduce and categorize different graph representations adopted and show how NLP tasks can be mapped onto graph-based problems and solved by graph neural network-based approaches in Sec. 21.2. After that, we will discuss two case studies. The first case study in Sec. 21.3 introduces graph-based text clustering and matching for hot events discovery and organization. The second one in Sec. 21.4 presents graph-based multi-hop machine reading comprehension. We then provide a synthesis about the important open problems of this subfield in Sec. 22.7. Finally, we conclude this chapter in Sec. 21.6.

Concurrently, a few very recent survey and tutorials (Wu et al, 2021c,b; Vashishth et al, 2019) aim to comprehensively introduce the historical and modern developments of machine learning (especially deep learning) on graphs for NLP. In addition, a recent released Graph4NLP library [1] is the first and an easy-to-use library at the intersection of Deep Learning on Graphs and Natural Language Processing. It provides both full implementations of state-of-the-art models for data scientists and also flexible interfaces to build customized models for researchers and developers with whole-pipeline support.

21.2 Modeling Text as Graphs

In this section, we will provide an overview of different graph representations in NLP. After that, we will discuss how different NLP tasks can be tackled from a graph perspective.

21.2.1 Graph Representations in Natural Language Processing

Various graph representations have been proposed for text modeling. Based on the different types of graph nodes and edges, a majority of existing works can be generalized into five categories: text graphs, syntactic graphs, semantic graphs, knowledge graphs, and hybrid graphs.

Text graphs use words, sentences, paragraphs, or documents as nodes and establish edges by word co-occurrence, location, or text similarities. Rousseau and Vazirgiannis (2013); Rousseau et al (2015) represented a document as graph-of-word, where nodes represent unique terms and directed edges represent co-occurrences between the terms within a fixed-size sliding window. Wang et al (2011) connected terms with syntactic dependencies. Schenker et al (2003) connected two words by

[1] Graph4NLP library can be accessed via this link https://github.com/graph4ai/graph4nlp.

a directed edge if one word immediately precedes another word in the document title, body, or link. The edges are categorized by the three different types of linking. Balinsky et al (2011); Mihalcea and Tarau (2004); Erkan and Radev (2004) connected sentences if they near to each other, share at least one common keyword, or the sentence similarity is above a threshold. Page et al (1999) connected web documents by hyperlinks. Putra and Tokunaga (2017) constructed directed graphs of sentences for text coherence evaluation. It utilized sentence similarities as weights and connects sentences with various constraints about sentence similarity or location. Text graphs can be established quickly, but they can not characterize the syntactic or semantic structure of sentences or documents.

Syntactic graphs (or trees) emphasize the syntactical dependencies between words in a sentence. Such structural representations of sentences are achieved by parsing, which constructs the syntactic structure of a sentence according to a formal grammar. Constituency parsing tree and dependency parsing graph are two types of syntactic representations of sentences that use different grammars (Jurafsky, 2000). Based on syntactic analysis, documents can also be structured. For example, Leskovec et al (2004) extracted subject-predicate-object triples from text based on syntactic analysis and merges them to form a directed graph. The graph was further normalized by utilizing WordNet (Miller, 1995) to merge triples belonging to the same semantic pattern.

While syntactic graphs show the grammatical structure of text pieces, semantic graphs aim to represent the meaning being conveyed. A model of semantics could help disambiguate the meaning of a sentence when multiple interpretations are valid. Abstract Meaning Representation (AMR) graphs (Banarescu et al, 2013) are rooted, labeled, directed, acyclic graphs (DAGs), comprising whole sentences. Sentences that are similar in meaning will be assigned the same AMR, even if they are not identically worded. In this way, AMR graphs abstract away from syntactic representations. The nodes in an AMR graph are AMR concepts, which are either English words, PropBank framesets (Kingsbury and Palmer, 2002), or special keywords. The edges are approximately 100 relations, including frame arguments following PropBank conventions, semantic relations, quantities, date-entities, lists, and so on.

Knowledge graphs (KGs) are graphs of data intended to accumulate and convey knowledge of the real world. The nodes of a KG represent entities of interest, and the edges represent relations between these entities (Hogan et al, 2020). Prominent examples of KGs include DBpedia (Bizer et al, 2009), Freebase (Bollacker et al, 2007), Wikidata (Vrandečić and Krötzsch, 2014) and YAGO (Hoffart et al, 2011), covering various domains. KGs are broadly applied for commercial use-cases, such as web search in Bing (Shrivastava, 2017) and Google (Singhal, 2012), commerce recommendation in Airbnb (Chang, 2018) and Amazon (Krishnan, 2018), and social networks like Facebook (Noy et al, 2019) and LinkedIn (He et al, 2016b). There are also graph representations that connect terms in a document to real-world entities or concepts based on KGs such as DBpedia (Bizer et al, 2009) and WordNet (Miller, 1995). For example, Hensman (2004) identifies the semantic roles in a sentence with

WordNet and VerbNet, and combines these semantic roles with a set of syntactic rules to construct a concept graph.

Hybrid graphs contain multiple types of nodes and edges to integrate heterogeneous information. In this way, the various text attributes and relations can be jointly utilized for NLP tasks. Rink et al (2010) utilized sentences as nodes and encodes lexical, syntactic, and semantic relations in edges. Jiang et al (2010) extracted tokens, syntactic structure nodes, semantic nodes and so on from each sentence and link them by different types of edges. Baker and Ellsworth (2017) built a sentence graph based on Frame Semantics and Construction Grammar.

21.2.2 Tackling Natural Language Processing Tasks from a Graph Perspective

Understanding natural language is essentially understanding different textual elements and their relationships. Therefore, we can tackle different NLP tasks from a graph perspective based on the different representations we have introduced. In recent years, many research works apply graph neural networks (Wu et al, 2021d) to solve NLP problems. A majority of them are actually solving the following problems: node classification, link prediction, graph classification, graph matching, community detection, graph-to-text generation, and reasoning over graphs.

For tasks focusing on assigning labels to words or phrases, they can be modeled as node classification. Cetoli et al (2017) showed that dependency trees play a positive role for named entity recognition by using a graph convolutional network (GCN) (Kipf and Welling, 2017b) to boost the results of a bidirectional LSTM. In (Gui et al, 2019), a GNN-based approach was proposed to alleviate the word ambiguity in Chinese NER. Lexicons are used to construct the graph and provide word-level features. Yao et al (2019) proposed a text classification method termed Text Graph Convolutional Networks. It builds a heterogeneous word document graph for a whole corpus and turns document classification into a node classification problem.

In addition to node classification, predicting the relationships between two elements is also an essential problem in NLP research, especially for knowledge graphs. Zhang and Chen (2018b) proposed a novel link prediction framework to simultaneously learn from local enclosing subgraphs, embeddings, and attributes based on graph neural networks. Rossi et al (2021) presented an extensive comparative analysis on link prediction models based on KG embeddings. They found that the graph structural features play paramount effects on the effectiveness of link prediction models. Guo et al (2019d) introduced the Attention Guided Graph Convolutional Networks (AGGCNs) for relation extraction tasks. The model operates directly on the full dependency trees and learns to distill the useful information from them in an end-to-end fashion.

Graph classification techniques are applied to text classification problems to utilize the intrinsic structure of texts. In (Peng et al, 2018), a graph-CNN based deep learning model was proposed for text classification. It first converts texts to graph-

of-words and then utilizes graph convolution operations to convolve the word graph. Huang et al (2019a); Zhang et al (2020d) proposed graph-based methods for text classification, where each text owns its structural graph and text level word interactions can be learned.

For NLP tasks involving a pair of text, graph matching techniques can be applied to incorporate the structural information of a text. Liu et al (2019a) proposed the Concept Interaction Graph to represent an article as a graph of concepts. It then matches a pair of articles by comparing the sentences that enclose the same concept node through a series of encoding techniques and aggregate the matching signals through a graph convolutional network. Haghighi et al (2005) represented sentences as directed graphs extracted from a dependency parser and develops a learned graph matching approach to approximating textual entailment. Xu et al (2019e) formulated the KB-alignment task as a graph matching problem, and proposed a graph attention-based approach. It first matches all entities in two KGs, and then jointly models the local matching information to derive a graph-level matching vector.

Community detection provides a means of coarse-graining the complex interactions or relations between nodes, which is suitable for text clustering problems. For example, Liu et al (2017a, 2020a) described a news content organization system at Tencent which discovers events from vast streams of breaking news and evolves news story structures in an online fashion. They constructed a keyword graph and applied community detection over it to perform coarse-grained keyword-based text clustering. After that, they further constructed a document graph for each coarse-grained clusters, and applied community detection again to get fine-grained event-level document clusters.

The task of graph-to-text generation aims at producing sentences that preserve the meaning of input graphs (Song et al, 2020b). Koncel-Kedziorski et al (2019) introduced a graph transforming encoder which can leverage the relational structure of knowledge graphs and generate text from them. Wang et al (2020k); Song et al (2018) proposed graph-to-sequence models (Graph Transformer) to generate natural language texts from AMR graphs. Alon et al (2019a) leveraged the syntactic structure of programming languages to encode source code and generate text.

Last but not least, reasoning over graphs plays a key role in multi-hop question answering (QA), knowledge-based QA, and conversational QA tasks. Ding et al (2019a) presented a framework CogQA to tackle multi-hop machine reading problem at scale. The reasoning process is organized as a cognitive graph, reaching entity-level explainability. Tu et al (2019) represented documents as a heterogeneous graph and employ GNN-based message passing algorithms to accumulate evidence on the proposed graph to solve the multi-hop reading comprehension problem across multiple documents. Fang et al (2020) created a hierarchical graph by constructing nodes on different levels of granularity (questions, paragraphs, sentences, entities), and proposed Hierarchical Graph Network (HGN) for multi-hop QA. Chen et al (2020n) dynamically constructed a question and conversation history aware context graph at each conversation turn and utilized a Recurrent Graph Neural Network and a flow mechanism to capture the conversational flow in a dialog.

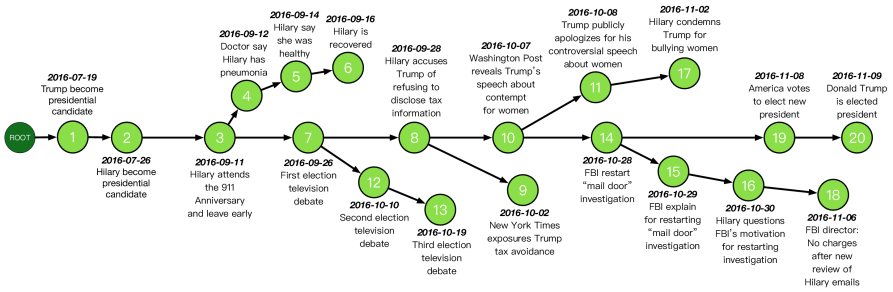

Fig. 21.1: The story tree of "2016 U.S. presidential election". Figure credit: Liu et al (2020a).

In the following, we will present two case studies to illustrate how graphs and graph neural networks can be applied to different NLP tasks with more details.

21.3 Case Study 1: Graph-based Text Clustering and Matching

In this case study, we will describe the Story Forest intelligent news organization system designed for fine-grained hot event discovery and organization from web-scale breaking news (Liu et al, 2017a, 2020a). Story Forest has been deployed in the Tencent QQ Browser, a mobile application that serves more than 110 million daily active users. Specifically, we will see how a number of graph representations are utilized for fine-grained document clustering and document pair matching and how GNN contributes to the system.

21.3.1 Graph-based Clustering for Hot Events Discovery and Organization

In the fast-paced modern society, tremendous volumes of news articles are constantly being generated by different media providers, leading to information explosion. In the meantime, the large quantities of daily news stories that can cover different subjects and contain redundant or overlapping data are becoming increasingly difficult for readers to digest. Many news app users feel that they are overwhelmed by extremely repetitive information about a variety of current hot events while still struggling to get information about the events in which they are genuinely interested. Besides, search engines conduct document retrieval on the basis of user-entered requests. They do not, however, provide users with a natural way to view trending topics or breaking news.

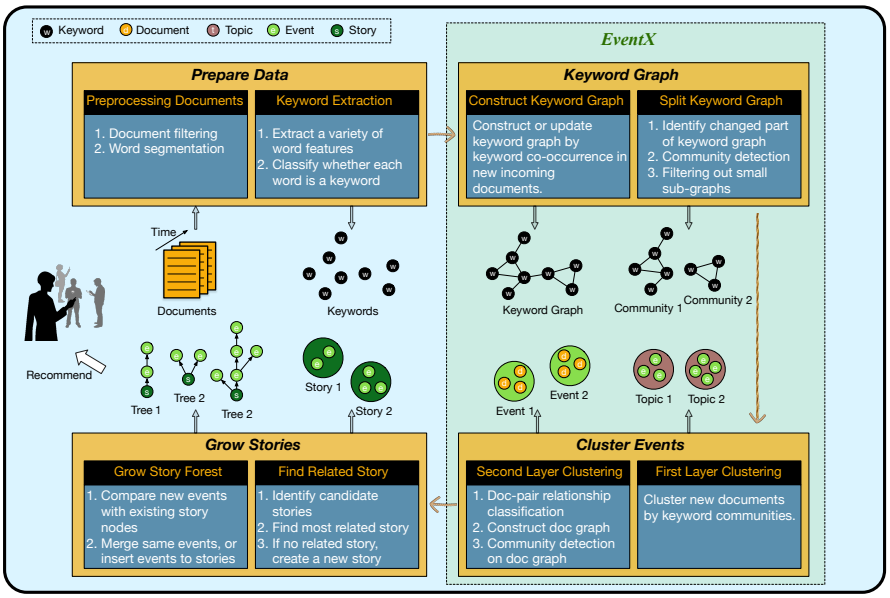

Fig. 21.2: An overview of the system architecture of Story Forest. Figure credit: Liu et al (2020a).

In (Liu et al, 2017a, 2020a), a novel news organization system named Story Forest was proposed to address the aforementioned challenges. The key idea of the Story Forest system is that, instead of providing users a list of web articles based on input queries, it proposes the concept of "event" and "story", and propose to organize tremendous of news articles into story trees to organize and track evolving hot events, revealing the relationships between them and reduce the redundancies. An event is a set of news articles reporting the same piece of real-world breaking news. And a story is a tree of related events that report a series of evolving real-world breaking news.

Figure 21.1 presents an example of a story tree, which showcases the story of "2016 U.S. presidential election". There are 20 nodes in the story tree. Each node indicates an event in the U.S. election in 2016, and each edge represents a temporal development relationship or a logical connection between two breaking news events. For example, event 1 is talking about Trump becomes a presidential candidate, and event 20 says Donald Trump is elected president. The index number on each node represents the event sequence over the timeline. The story tree contains 6 paths, where the main path $1 \rightarrow 20$ captures the process of the presidential election, the branch $3 \rightarrow 6$ describes Hilary's health conditions, the branch $7 \rightarrow 13$ is focusing on the television debates, $14 \rightarrow 18$ are about "mail door" investigation, etc. As we can see, users can easily understand the logic of news reports and learn the key facts quickly by modeling the evolutionary and logical structure of a story into a story tree.

The story trees are constructed from web-scale news articles by the Story Forest system. The system's architecture is shown in Fig. 21.2. It consists primarily of four components: preprocessing, keyword graph construction, clustering documents to events, and growing story trees with events. The overall process is split into eight stages. First, a range of NLP and machine learning tools will be used to process the input news document stream, including document filtering and word segmentation. Then the system extracts keywords, construct/update the co-occurrence graph of keywords, and divide the graph into sub-graphs. After that, it utilizes *EventX*, a graph-based fine-grained clustering algorithm to cluster documents into fine-grained events. Finally, the story trees (formed previously) are updated by either inserting each discovered event into an existing story tree at the right place or creating a new story tree if the event does not belong to any current story.

We can observe from Fig. 21.2 that a variety of text graphs are utilized in the Story Forest system. Specifically, the *EventX* clustering algorithm is based on two types of text graphs: keyword co-occurrence graph and document relationship graph. The keyword co-occurrence graph connects two keywords if they co-occurred for more than n times in a news corpus, where n is a hyperparameter. On the other hand, the document relationship graph connects document pairs based on whether two documents are talking about the same event. Based on such two types of text graphs, EventX can accurately extract fine-grained document clusters, where each cluster contains a set of documents that focus on the same event.

In particular, EventX performs two-layer graph-based clustering to extract events. The first layer performs community detection over the constructed keyword co-occurrence graph to split it into sub-graphs, where each sub-graph the keywords for a specific topic. The intuition for this step is that keywords related to a common topic usually will frequently appear in documents belonging to that topic. For example, documents belonging to the topic "2016 U.S. presidential election" will often mention keywords such as "Donald Trump", "Hillary Clinton", "election", and so on. Therefore, highly correlated keywords will be linked to each other and form dense subgraphs, whereas keywords that are not highly related will have sparse or no links. The goal here is to extract dense keyword subgraphs linked to various topics. After obtaining the keyword subgraphs (or communities), we can assign each document to its most correlated keyword subgraph by calculating their TF-IDF similarity. At this point, we have grouped documents by topics in the first layer clustering.

In the second layer, EventX constructs a document relationship graph for each topic obtained in the first layer. Specifically, a binary classifier will be applied to each pair of documents in a topic to detect whether two documents are talking about the same event. If yes, we connect the pair of documents. In this way, the set of documents in a topic turn into a document relationship graph. After that, the same community detection algorithm in the first layer will be applied to the document relationship graph, splitting it into sub-graphs where each sub-graph now represents a fine-grained event instead of a coarse-grained topic. Since the number of news articles belonging to each topic is significantly less after the first-layer document clustering, the graph-based clustering on the second layer is highly efficient, making it applicable for real-world applications. After extracting fine-grained events, we can

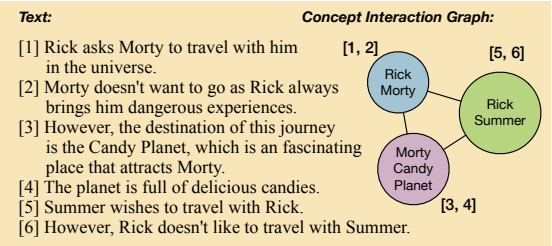

Fig. 21.3 An example to show a piece of text and its Concept Interaction Graph representation. Figure credit: Liu et al (2019a)

update the story trees by inserting an event to its related story or creating a new story tree if it doesn't belong to any existing stories. We refer to (Liu et al, 2020a) for more details about the Story Forest system.

21.3.2 Long Document Matching with Graph Decomposition and Convolution

During the construction of the document relationship graph in the Story Forest system, a fundamental problem is determining whether two news articles are talking about the same event. It is a problem of semantic matching, which is a core research problem that lies at the core of many NLP applications, including search engines, recommender systems, news systems, etc. However, previous research about semantic matching is mainly designed for matching sentence pairs (Wan et al, 2016; Pang et al, 2016), e.g., for paraphrase identification, answer selection in question-answering, and so on. Due to the long length of news articles, such methods are not suitable and do not perform well on document matching (Liu et al, 2019a).

To solve this challenge, Liu et al (2019a) presented a divide-and-conquer strategy to align a pair of documents and shift deep text comprehension away from the currently dominant sequential modeling of language elements and toward a new level of graphical document representation that is better suited to longer articles. Specifically, Liu et al (2019a) proposed the Concept Interaction Graph (CIG) as a way to view a document as a weighted graph of concepts, with each concept node being either a keyword or a group of closely related keywords. Furthermore, two concept nodes will be connected by a weighted edge which indicates their interaction strength.

As a toy example, Fig. 21.3 shows how to convert a document into a Concept Interaction Graph (CIG). First, we extract keywords such as *Rick*, *Morty*, and *Summer* from the document using standard keyword extraction algorithms, e.g., TextRank (Mihalcea and Tarau, 2004). Second, similar to what we have done in the Story Forest system, we can group keywords into sub-graphs by community detection. Each keyword community turns into a "concept" in the document. After extracting concepts, we attach each sentence in the document to its most related concept node by calculating the similarities between a sentence and each concept. In Fig. 21.3, sen-

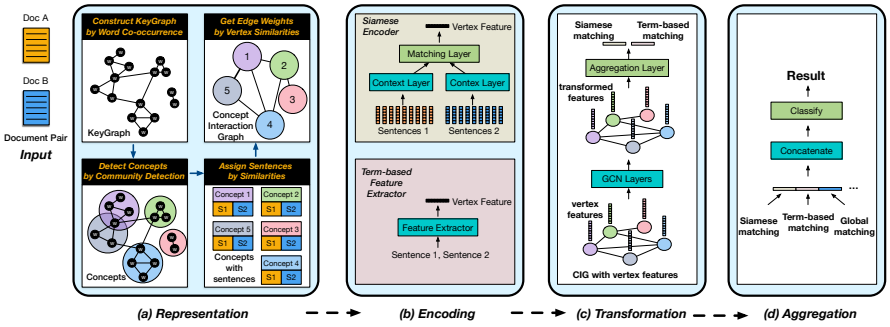

Fig. 21.4: An overview of our approach for constructing the Concept Interaction Graph (CIG) from a pair of documents and classifying it by Graph Convolutional Networks. Figure credit: Liu et al (2019a).

tences 5 and 6 are mainly talking about the relationship between *Rick* and *Summer*, and are thus attached to the concept (*Rick*, *Summer*). Similarly, we can attach other sentences to nodes, decomposing the content of a document into a number of concepts. To construct edges, we represent each node's sentence set as a concatenation of the sentences attached to it and measure the edge weight between any two nodes as the TF-IDF similarity between their sentence sets to create edges that show the correlation between different concepts. An edge will be removed if its weight is below a threshold. For a pair of documents, the process of converting them into a CIG is similar. The only differences are that the keywords are from both documents, and each concept node will have two sets of sentences from the two documents. As a result, we have represented the original document (or document pair) with a graph of key concepts, each with a (or a pair of) sentence subset(s), as well as the interaction topology among them.

The CIG representation of a document pair decomposes its content into multiple parts. Next, we need to match the two documents based on such representation. Fig. 21.4 illustrates the process of matching a pair of long documents. The matching process consists of four steps: a) preprocessing the input document pair and transform it into a CIG; b) matching the sentences from two documents over each node to get local matching features; c) structurally transforming local matching features by graph convolutional layers; and d) aggregating all the local matching features to get the final result.

Specifically, for the local matching on each concept node, the inputs are the two sets of sentences from two documents. As each node only contains a small portion of the document sentences, the long text matching problems transform into short text matching on a number of concept nodes. In (Liu et al, 2019a), two different matching models are utilized: i) similarity-based matching, which calculate a variety of text similarities between two set of sentences; ii) Siamese matching, which utilizes a Siamese neural network (Mueller and Thyagarajan, 2016) to encode the

two sentence sets and get a local matching vector. After getting local matching results, the next question is: how to get an overall matching score? Liu et al (2019a) aggregates the local matching vectors into a final matching score for the pair of articles by utilizing the ability of the graph convolutional network filters (Kipf and Welling, 2017b) to capture the patterns exhibited in the CIG at multiple scales. In particular, the local matching vectors of the concept nodes are transformed by multi-layer GCN layers to take the interaction structure between nodes (or concepts in two documents) into consideration. After getting the transformed feature vectors, they are aggregated by mean pooling to get a global matching vector. Finally, the global matching vector will be fed into a classifier (e.g., a feed-forward neural network) to get the final matching label or score. The local matching module, global aggregation module, and the final classification module are trained end-to-end.

In (Liu et al, 2019a), extensive evaluations were performed to test the performance of the proposed approach for document matching. A key discovery made by (Liu et al, 2019a) is that the graph convolution operation significantly improves the performance of matching, demonstrating the effect of applying graph neural networks to the proposed text graph representation. The structural transformation on the matching vectors via GCN can efficiently capture the semantic interactions between sentences, and the transformed matching vectors better capture the semantic distance over each concept node by integrating the information of its neighbor nodes.

21.4 Case Study 2: Graph-based Multi-Hop Reading Comprehension

In this case study, we further introduce how graph neural networks can be applied to machine reading comprehension in NLP. Machine reading comprehension (MRC) aims to teach machines to read and understand unstructured text like a human. It is a challenging task in artificial intelligence and has great potential in various enterprise applications. We will see that by representing text as a graph and applying graph neural networks to it, we can mimic the reasoning process of human beings and achieve significant improvements for MRC tasks.

Suppose we have access to a Wikipedia search engine, which can be utilized to retrieve the introductory paragraph $para[x]$ of an entity x. How can we answer the question "Who is the director of the 2003 film which has scenes in it filmed at the Quality Cafe in Los Angeles?" with the search engine? Naturally, we will start with pay attention to related entities such as "Quality Cafe", look up relevant introductions through Wikipedia, and quickly locate "Old School" and "Gone in 60 Seconds" when it comes to Hollywood movies. By continuing to inquire about the introduction of the two movies, we further found their director. The last step is to determine which director it is. This requires us to analyze the semantics and qualifiers of the sentence. After knowing that the movie is in 2003, we can make the final judgment: "Todd Phillips" is the answer we want. Figure 21.5 illustrates such

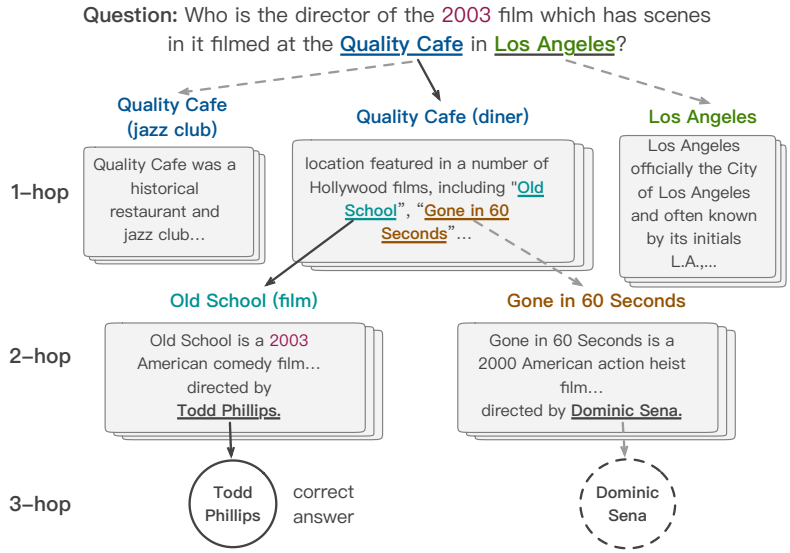

Fig. 21.5: An example of cognitive graph for multi-hop QA. Each *hop node* corresponds to an entity (e.g., "Los Angeles") followed by its introductory paragraph. The circles mean *ans nodes*, answer candidates to the question. Cognitive graph mimics human reasoning process. Edges are built when calling an entity to "mind". The solid black edges are the correct reasoning path. Figure credit: Ding et al (2019a).

process. Answering the aforementioned question requires multi-hop reasoning over different information, that is so-called multi-hop question answering.

In fact, "pay attention to related entities quickly" and "analyze the meaning of sentences for inference" are two different thinking processes. In cognition, the well-known "dual process theory" (Kahneman, 2011) believes that human cognition is divided into two systems. **System 1** is an implicit, unconscious and intuitive thinking system. Its operation relies on experience and association. **System 2** performs explicit, conscious and controllable reasoning process. This system uses knowledge in working memory to perform slow but reliable logical reasoning. System 2 is the embodiment of human advanced intelligence.

Guided by the dual process theory, the Cognitive Graph QA (CogQA) framework was proposed in (Ding et al, 2019a). It adopts a directed graph structure, named cognitive graph, to perform step-by-step deduction and exploration in the cognitive process of multi-hop question answering. Figure 21.5 presents the cognitive graph for answering the previously mentioned question. Denote the graph as $\mathscr{G}$, each node in $\mathscr{G}$ represents an entity or possible answer x, also interchangeably denoted as node x. The solid black edges are the correct reasoning path to answer the question. The cognitive graph is constructed by an extraction module that acts like System 1. It

takes the introductory paragraph $para[x]$ of entity x as input, and outputs answer candidates (i.e., *ans nodes*) and useful next-hop entities (i.e., *hop nodes*) from the paragraph. These new nodes gradually expand $\mathscr{G}$, forming an explicit graph structure for System 2 reasoning module. During the expansion of $\mathscr{G}$, the new nodes or existing nodes with new incoming edges bring new *clue* about the answer. Such nodes are referred as *frontier nodes*. For *clue*, it is a form-flexible concept, referring to information from predecessors for guiding System 1 to better extract spans. To perform neural network-based reasoning over $\mathscr{G}$ instead of rule-based, System 1 also summarizes $para[x]$ into an initial hidden representation vector when extracting spans, and System 2 updates all paragraphs' hidden vectors X based on graph structure as reasoning results for downstream prediction.

The procedure of the framework CogQA is as follows. First, the cognitive graph $\mathscr{G}$ is initialized with the entities mentioned in the input question Q, and the entities are marked as initial frontier nodes. After initialization, a node x is popped from frontier nodes, and then a two-stage iterative process is conducted with two models $\mathscr{S}_1$ and $\mathscr{S}_2$ mimicking System 1 and System 2, respectively. In the first stage, the System 1 module in CoQA extracts question-relevant entities, answers candidates from paragraphs, and encodes their semantic information. Extracted entities are organized as a cognitive graph, which resembles the working memory. Specifically, given x, CogQA collects $clues[x, \mathscr{G}]$ from predecessor nodes of x, where the *clues* can be sentences where x is mentioned. It further fetches introductory paragraph $para[x]$ in Wikipedia database $\mathscr{W}$ if any. After that, $\mathscr{S}_1$ generates $sem[x, Q, clues]$, which is the initial X_x (i.e., the embedding of x). If x is a *hop node*, then $\mathscr{S}_1$ finds hop (e.g., entities) and answer spans in $para[x]$. For each hop span y, if $y \notin \mathscr{G}$ and $y \in \mathscr{W}$, then create a a new hop node for y and add it to $\mathscr{G}$. If $y \in \mathscr{G}$ but $edge(x, y) \notin \mathscr{G}$, then add a new edge (x, y) to $\mathscr{G}$ and mark node y as a frontier node, as it needs to be revisited with new information. For each answer span y, a new answer node y and edge (x, y) will be added to $\mathscr{G}$. In the second stage, System 2 conducts the reasoning procedure over the graph and collects clues to guide System 1 to better extract next-hop entities. In particular, the hidden representation X of all paragraphs will be updated by $\mathscr{S}_2$. The above process is iterated until there is no frontier node in the cognitive graph (i.e., all possible answers are found) or the graph is large enough. Then the final answer is chosen with a predictor $\mathscr{F}$ based on the reasoning results X from System 2.

The CogQA framework can be implemented as the system in Fig. 21.6. It utilizes BERT (Devlin et al. 2019) as System 1 and GNN as System 2. For clues $clues[x, \mathscr{G}]$, they are the sentences in paragraphs of x's predecessor nodes, from which x is extracted. We can observe from Fig. 21.6 that the input to BERT is the concatenation of the question, the clues passed from predecessor nodes, and the introductory paragraph of x. Based on these inputs, BERT outputs hop spans and answer spans, as well as uses the output at position 0 as $sem[x, Q, clues]$.

For System 2, CogQA utilizes a variant of GNN to update the hidden representations of all nodes. For each node x, its initial representation $X_x \in \mathbb{R}^h$ is the semantic vector $sem[x, Q, clues]$ from System 1 (i.e., BERT). The updating formula of the GNN layers are as follows:

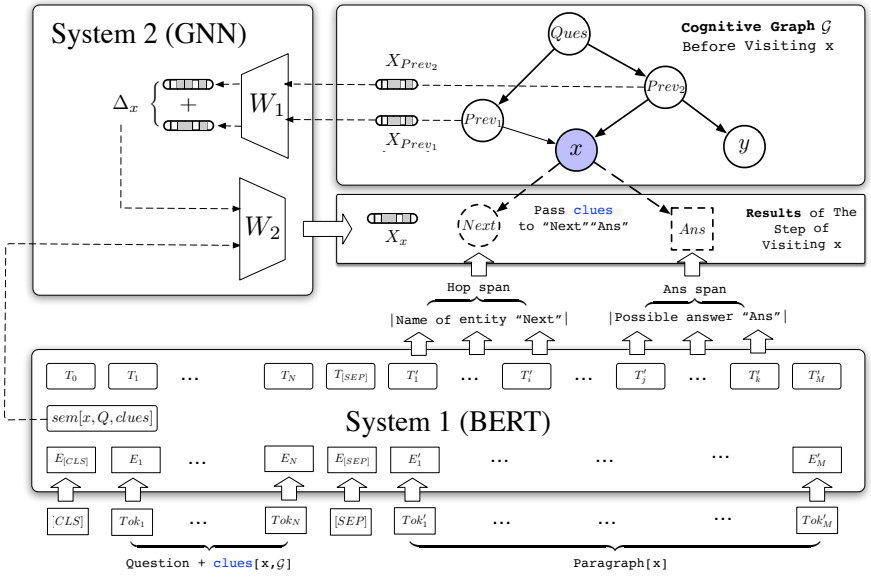

Fig. 21.6: Overview of CogQA implementation. When visiting the node x, System 1 generates new hop and answer nodes based on the $clues[x, \mathcal{G}]$ discovered by System 2. It also creates the inital representation $sem[x, Q, clues]$, based on which the GNN in System 2 updates the hidden representations X_x. Figure credit: Ding et al (2019a).

$$\Delta = \sigma((AD^{-1})^{\top}\sigma(XW_1)) \tag{21.1}$$

$$X' = \sigma(XW_2 + \Delta) \tag{21.2}$$

where X' is the new hidden representations after a propagation step of GNN. $W_1, W_2 \in \mathbb{R}^{h \times h}$ are weight matrices, σ is the activation function. $\Delta \in \mathbb{R}^{n \times h}$ are aggregated vectors passed from neighbors in the propagation. A is the adjacent matrix of $\mathcal{G}$. It is column-normalized to AD^{-1}, where D is the degree matrix of $\mathcal{G}$. By left multiplying the transformed hidden vector $\sigma(XW_1)$ with $(AD^{-1})^{\top}$, the GNN performs a localized spectral filtering. In the iterative step of visiting frontier node x, its hidden representation X_x is updated following the above equations.

Finally, a two-layer fully connected network (FCN) is utilized to serve as predictor $\mathcal{F}$:

$$answer = \underset{\text{answer node } x}{\arg\max} \ \mathcal{F}(X_x) \tag{21.3}$$

In this way, one answer candidate can be selected as the final answer. In the HotpotQA dataset (Yang et al, 2018b), there are also questions that aim to compare a certain property of entity x and y. Such questions are regarded as binary classification with input $X_x - X_y$ and solved by another identical FCNs.

The cognitive graph structure in the CogQA framework offers ordered and entity-level explainability and suits for relational reasoning, owing to the explicit reasoning paths in it. Aside from simple paths, it can also clearly display joint or loopy reasoning processes, where new predecessors might bring new clues about the answer. As we can see, by modeling the context information as a cognitive graph and applying GNN to such representation, we can mimic the dual process of human perception and reasoning and achieve excellent performance on multi-hop machine reading comprehension tasks, as demonstrated in (Ding et al, 2019a).

21.5 Future Directions

Applying graph neural networks to NLP tasks with suitable graph representations for text can bring significant benefits, as we have discussed and shown through the case studies. Although GNNs have achieved outstanding performance in many tasks, including text clustering, classification, generation, machine reading comprehension and so on, there are still numerous open problems to solve at the moment to better understand human language with graph-based representations and models. In particular, here we categorize and discuss the open problems or future directions for graph-based NLP in terms of five aspects: model design of GNNs, data representation learning, multi-task relationship modeling, world model, and learning paradigm.

Although several GNN models are applicable to NLP tasks, only a small subset of them is explored for model design. More advanced GNN models can be utilized or improved to handle the scale, depth, dynamics, heterogeneity, and explainability of natural language texts. First, scaling GNNs to large graphs helps to utilize resources such as large-scale knowledge graphs better. Second, most GNN architectures are shallow, and the performance drops after two to three layers. Design deeper GNNs enables node representation learning with information from larger and more adaptive receptive fields (Liu et al, 2020c). Third, we can utilize dynamic graphs to model the evolving or temporal phenomenons in texts, e.g., the development of stories or events. Correspondingly, dynamic or temporal GNNs (Skarding et al, 2020) can help capture the dynamic nature in specific NLP tasks. Forth, the syntactic, semantic, as well as knowledge graphs in NLP are essentially heterogeneous graphs. Developing heterogeneous GNNs (Wang et al, 2019i; Zhang et al, 2019b) can help better utilizing the various nodes and edge information in text and understanding its semantic. Last but not least, the need for improved explainability, interpretability, and trust of AI systems in general demands principled methodologies. One way is using GNNs as a model of neural-symbolic computing and reasoning (Lamb et al, 2020), as the data structure and reasoning process can be naturally captured by graphs.

For data representations, most existing GNNs can only learn from input when a graph-structure of input data is available. However, real-world graphs are often noisy and incomplete or might not be available at all. Designing effective models and algorithms to automatically learn the relational structure in input data with lim-

ited structured inductive biases can efficiently solve this problem. Instead of manually designing specific graph representations of data for different applications, we can enable models to automatically identify the implicit, high-order, or even casual relationships between input data points, and learn the graph structure and representations of inputs. To achieve these, recent research on graph pooling (Lee et al, 2019b), graph transformers (Yun et al, 2019), and hypergraph neural networks (Feng et al, 2019c) can be applied and further explored.

Multi-task learning (MTL) in deep neural networks for NLP has recently received increasing interest as it has the potential to efficiently regularize models and to reduce the need for labeled data (Bingel and Søgaard, 2017). We can marriage the representation power of graph structures with multi-task learning to integrate diverse input data, such as images, text pieces, and knowledge bases, and jointly learn a unified and structured representation for various tasks. Furthermore, we can learn the relationships or correlations between different tasks and exploit the learned relationship for curriculum learning to accelerate the convergence rate for model training. Finally, with the unified graph representation and integration of different data, as well as the joint and curriculum learning of different tasks, NLP or AI systems will gain the ability to continually acquire, fine-tune, and transfer knowledge and skills throughout their lifespan.

Grounded language learning or acquisition (Matuszek, 2018; Hermann et al, 2017) is another trending research topic that aims at learning the meaning of language as it applies to the physical world. Intuitively, language can be better learned when presented and interpreted in the context of the world it pertains to. It has been demonstrated that GNNs can efficiently capture joint dependencies between different elements in the world (Li et al, 2017e). Besides, they can also efficiently utilize the rich information in multiple modalities of the world to help understand the meaning of scene texts (Gao et al, 2020a). Therefore, representing the world or environment with graphs and GNNs to improve the understanding of languages deserves more research endeavors.

Lastly, research about self-supervised pre-training for GNNs is also attracting more attention. Self-supervised representation learning leverages input data itself as supervision and benefits almost all types of downstream tasks (Liu et al, 2020f). Numerous successful self-supervised pre-training strategies, such as BERT (Devlin et al, 2019) and GPT (Radford et al, 2018) have been developed to tackle a variety of language tasks. For graph learning, when task-specific labeled data is extremely scarce, or the graphs in the training set are structurally very different from graphs in the test set, pre-training GNNs can serve as an efficient approach for transfer learning on graph-structured data (Hu et al, 2020c).

21.6 Conclusions

Over the past few years, graph neural networks have become powerful and practical tools for a variety of problems that can be modeled by graphs. In this chapter, we

did a comprehensive overview of combining graph representations and graph neural networks in NLP tasks. We introduced the motivation of applying graph representations and GNNs to NLP problems through the developing history of NLP research. After that, we provided a brief overview of various graph representations in NLP, as well as discussed how to tackle different NLP tasks from a graph perspective. To illustrate how graphs and GNNs are applied in NLP applications with more details, we presented two case studies related to graph-based hot event discovery and multi-hop machine reading comprehension. Finally, we categorized and discussed several frontier research and open problems for graph-based NLP.

Editor's Notes: Graph-based methods for Natural Language Processing have been long studied over the last two decades. Indeed, the human language is high-level symbol and thus there are rich hidden structural information beyond the original simple text sequence. In order to make full use of GNNs for NLP, graph structure learning techniques in Chapter 14 and GNN Methods in Chapter 4 serve as the two fundamental building blocks. Meanwhile, GNN Scalability in Chapter 6, Heterogeneous GNNs in Chapter 16, GNN Robustness in Chapter 8, and so on are also highly important for developing an effective and efficient approach with GNNs for various NLP applications. This chapter is also highly correlated with the Chapter 20 (GNN for CV) since vision and language is a fast-growing research area and multi-modality data is widely used today.