10
Graph Neural Networks in Natural Language Processing

10.1 Introduction

Graphs have been extensively utilized in natural language process (NLP) to represent linguistic structures. The constituency-based parse trees represent phrase structures for a given sentence. The syntactic dependency trees encode syntactic relations in terms of tree structures (Jurafsky and Martin, n.d.). Abstract meaning representation (AMR) denotes semantic meanings of sentences as rooted and labeled graphs that are easy for the program to traverse (Banarescu et al., 2013). These graph representations of natural languages carry rich semantic and/or syntactic information in an explicit structural way. Graph neural networks (GNNs) have been adopted by various NLP tasks where graphs are involved. These graphs include those mentioned above and also other graphs designed specifically for particular tasks. Specifically, GNNs have been utilized to enhance many NLP tasks such as semantic role labeling (Marcheggiani and Titov, 2017), (multi-hop) question answering (QA) (De Cao et al., 2019; Cao et al., 2019; Song et al., 2018a; Tu et al., 2019), relation extraction (Zhang et al., 2018c; Fu et al., 2019; Guo et al., 2019; Zhu et al., 2019b; Sahu et al., 2019; Sun et al., 2019a; Zhang et al., 2019d), neural machine translation (Marcheggiani et al., 2018; Beck et al., 2018), and graph to sequence learning (Cohen, 2019; Song et al., 2018b; Xu et al., 2018b). Furthermore, knowledge graphs, which encode multi-relational information in terms of graphs, are widely adopted by NLP tasks. There are also many works (Hamaguchi et al., 2017; Schlichtkrull et al., 2018; Nathani et al., 2019; Shang et al., 2019a; Wang et al., 2019c; Xu et al., 2019a) generalizing GNN models to knowledge graphs. In this chapter, we take semantic role labeling, neural machine translation, relation extraction, question answering, and graph to sequence learning as examples to demonstrate how graph neural networks can

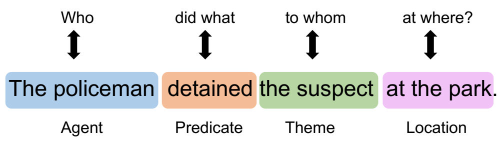


Figure 10.1 An illustrative sentence with semantic labels.

be applied to NLP tasks. We also introduce the graph neural network models designed for knowledge graphs.

10.2 Semantic Role Labeling

In (Marcheggiani and Titov, 2017), GNNs are utilized on syntactic dependency trees to incorporate syntactic information to improve the performance of Semantic Role Labeling (SRL). It is among the first to show that graph neural network models are effective on NLP tasks. In this section, we first describe the task of Semantic Role Labeling (SRL) and then introduce how GNNs can be leveraged for this task.

Semantic Role Labeling aims to discover the latent predicate argument structure of a sentence, which can be informally regarded as the task of discovering "who did what to whom at where?". For example, a sentence with semantic labels is shown in Figure 10.1 where the word "detained" is the predicate, "the policeman" and "the suspect" are its two arguments with different labels. More formally, the task of SRL involves the following steps: 1) detecting the predicates such as "detained" in Figure 10.1; and 2) identifying the arguments and labeling them with semantic roles, i.e., "the policeman" is the agent while "the suspect" is the theme. In (Marcheggiani and Titov, 2017), the studied SRL problem (on CoNLL-2009 benchmark) is simplified a little bit, where the predicate is given in the test time (e.g., we know that "detained" is the predicate in the example shown in Figure 10.1), hence no predicate detection is needed. The remaining task is to identify the arguments of the given predicate and label them with semantic roles. It can be treated as a sequence labeling task. In detail, the semantic role labeling model is asked to label all the arguments of the given predicate with their corresponding labels and label "NULL" for all the non-argument elements.

To tackle this problem, a Bi-directional LSTM (Bi-LSTM) encoder is adopted by (Marcheggiani and Titov, 2017) to learn context-aware word representations. These learned word representations are later utilized to label each of the elements in the sequence. We denote a sentence as $[w_0, \ldots, w_n]$, where each word w_i in the sequence is associated with an input representation $\mathbf{x}_i$. The input representation consists of four components: 1) a randomly initialized embedding; 2) a pre-trained word embedding; 3) a randomly initialized embedding for its corresponding part-of-speech tag; and 4) a randomly initialized lemma embedding, which is active only when the word is a predicate. These four embeddings are concatenated to form the input representation $\mathbf{x}_i$ for each word w_i. Three of the embeddings except the pre-trained embedding are updated during the training. The sequence $[\mathbf{x}_0, \ldots, \mathbf{x}_n]$ is then utilized as the input for the Bi-LSTM (Goldberg, 2016). Specifically, the Bi-LSTM model consists of two LSTMs with one dealing with the input sequence for the forward pass while the other handling the sequence for the backward pass. The operations of a single LSTM unit is introduced in Section 3.4.2. In the following, we abuse the notation a little bit to use LSTM() to denote the process of dealing a sequence input with LSTM. The process of the forward and backward LSTM can be denoted as:

$$[\mathbf{x}_0^f, \ldots, \mathbf{x}_n^f] = \text{LSTM}^f([\mathbf{x_0}, \ldots, \mathbf{x}_n]),$$
$$[\mathbf{x}_0^b, \ldots, \mathbf{x}_n^b] = \text{LSTM}^b([\mathbf{x_n}, \ldots, \mathbf{x}_0]),$$

where LSTM^f denotes the forward LSTM, which captures the left context for each word, while LSTM^b denotes the backward LSTM that captures the right context for each word. Note that, $\mathbf{x}_i^b$ is the output representation from LSTM^b for the word w_{n-i}. The outputs of the two LSTMs are concatenated as the output of the Bi-LSTM, which captures the context information from both directions as:

$$[\mathbf{x}_0^{\text{bi}}, \ldots, \mathbf{x}_n^{\text{bi}}] = \text{Bi-LSTM}([\mathbf{x_0}, \ldots, \mathbf{x}_n]),$$

where $\mathbf{x}_i^{\text{bi}}$ is the concatenation of $\mathbf{x}_i^f$ and $\mathbf{x}_{n-i}^b$. With the output of Bi-LSTM, the labeling task is treated as a classification problem for each candidate word with the semantic labels and “NULL” as labels. Specifically, the input of the classifier is the concatenation of the output representations from the Bi-LSTM for the candidate word $\mathbf{x}_c^{\text{bi}}$ and for the predicate $\mathbf{x}_p^{\text{bi}}$.

To enhance the algorithm described above, syntactic structure information is incorporated by utilizing graph neural network models on syntactic dependency trees (Marcheggiani and Titov, 2017). In detail, the aggregation process in the graph neural network model is generalized to incorporate directed labeled edges such that it can be applied to syntactic dependency trees. To incorporate the sentence’s syntactic information, the output of the Bi-LSTM layer is employed as the input of the graph neural network model. Then, the output of

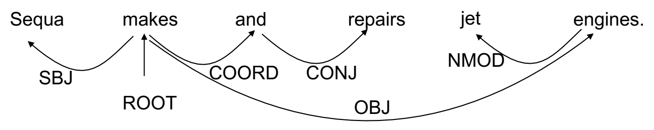


Figure 10.2 The dependency tree of the sentence "Sequa makes and repairs jet engines."

the graph neural network model is used as the input for the linear classifier described above. Next, we first briefly introduce syntactic dependency trees and then describe how the graph neural network model is modified for syntactic dependency trees.

A syntactic dependency tree is a directed labeled tree encoding the syntactic dependencies in a given sentence. Specifically, the words in the sentence are treated as the nodes for the dependency tree while the directed edges describe the syntactic dependency between them. The edges are labeled with various dependency relations such as "Subject" (SBJ) and "Direct Object" (DOBJ). As an illustrative example, the dependency tree of the sentence "Sequa makes and repairs jet engines." is shown in Figure 10.2, where "Sequa" is the subject of the verb "makes" and "engines" is the objective of "makes". As the edges are directed and labeled in the dependency tree, to adopt the graph neural network model to incorporate the direction and label information in the edge, the following generalized graph filtering operator (for the l-th layer) is proposed in (Marcheggiani and Titov, 2017):

$$\mathbf{F}_i^{(l)} = \sigma\left(\sum_{v_j \in \mathcal{N}(v_i)} \mathbf{F}_j^{(l-1)} \mathbf{\Theta}_{dir(i,j)}^{(l-1)} + \mathbf{b}_{lab(i,j)}\right), \tag{10.1}$$

where $\mathcal{N}(v_i)$ consists of both in-going and out-going neighbors of node v_i, $dir(i, j) \in \{\text{in-going}, \text{out-going}\}$ denotes the direction of the edge (v_i, v_j) in terms of the center node v_i, $\mathbf{\Theta}_{dir(i,j)}^{(l-1)}$ are the parameters shared by the edges that have the same direction as (v_i, v_j) and $\mathbf{b}_{lab(i,j)}$ is a bias term to incorporate the label information on the edge with $lab(i, j)$ denoting the dependency relation of (v_i, v_j). The filter described in Eq. (10.1) is utilized to build a graph neural network model with L layers for the SRL task.

10.3 Neural Machine Translation

Machine translation is an essential task in natural language processing. Modern machine translation models usually take the form of encoder-decoder. The encoder takes a sequence of words in the source language as input and outputs a representation for each word in the sequence. Then the decoder, relying on the representations from the encoder, outputs a translation (or a sequence of words in the target language). Both the encoder and decoder are usually modeled with recurrent neural networks or their variants. For example, the Bi-LSTM introduced in Section 10.2 is a popular choice for the encoder while RNN models equipped with the attention mechanism (Bahdanau et al., 2014) is the popular choice for the decoder. In (Marcheggiani et al., 2018), to incorporate the syntactic structure information in the sentence to enhance the performance of machine translation, the same strategy that is introduced in Section 10.2 is adopted to design the encoder. The decoder keeps the same as the traditional model, i.e., the attention-based RNN model. Next, we briefly describe the encoder, as we have already introduced it in Section 10.2. Specifically, a Bi-LSTM model is first utilized for encoding the sequence. These representations from Bi-LSTM are then served as the input for a graph neural network model on the syntactic dependency tree. The formulation of a single graph filtering operation of the graph neural network model is shown in Eq. (10.1). The output of the graph neural network model is then leveraged as the input for the decoder (Bastings et al., 2017).

10.4 Relation Extraction

Graph Neural Networks have also been applied to the relation extraction (RE) task (Zhang et al., 2018c; Fu et al., 2019; Guo et al., 2019; Zhu et al., 2019b; Sahu et al., 2019; Sun et al., 2019a; Zhang et al., 2019d). Specifically, the works (Zhang et al., 2018c; Fu et al., 2019; Guo et al., 2019) adopt and/or modify the graph neural network model (i.e., Eq. (10.1)) in (Marcheggiani and Titov, 2017) to incorporate the syntactic information for the task of relation extraction. The first work applying graph neural networks to RE is introduced in (Zhang et al., 2018c). In this section, we briefly describe the task of RE and then use the model in (Zhang et al., 2018c) as an example to demonstrate how graph neural networks can be adopted to RE.

The task of relation extraction is to discern whether a relation exists between two entities (i.e., *subject* and *object*) in a sentence. More formally, it can be defined as follows. Let $\mathcal{W} = [w_1, \ldots, w_n]$ denote a sentence, where w_i is the i-th token in the sentence. An entity is a span consisting of consecutive words in the sentence. Specifically, a subject entity, which consists of a series of consecutive words, can be represented as $\mathcal{W}_s = [w_{s1} : w_{s2}]$. Similarly, an object entity can be expressed as $\mathcal{W}_o = [w_{o1} : w_{o2}]$. The goal of relation extraction is to predict the relation for the subject entity $\mathcal{W}_s$ and the object entity $\mathcal{W}_o$ given the sentence $\mathcal{W}$, where $\mathcal{W}_s$ and $\mathcal{W}_o$ are assumed to be given. The relation is from a predefined set $\mathcal{R}$, which also includes a special relation "no relation" indicating that there is no relation between these two entities. The problem of relation extraction is treated as a classification problem in (Zhang et al., 2018c). The input is the concatenation of the representations of the sentence $\mathcal{W}$, the subject entity $\mathcal{W}_s$ and the object entity $\mathcal{W}_o$. The output labels are the relations in $\mathcal{R}$. Specifically, the relation prediction for a pair of entities is through a feed-forward neural network (FFNN) with parameters Θ_{FFNN} as shown below:

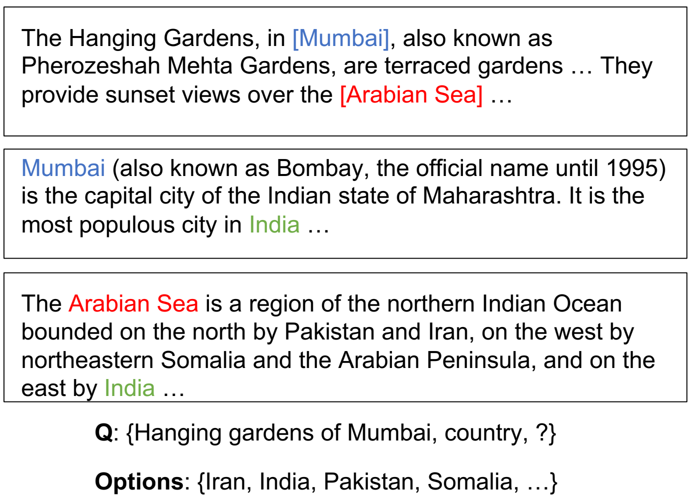
$$\mathbf{p} = \text{softmax}([\mathbf{F}_{sent}, \mathbf{F}_s, \mathbf{F}_o]\boldsymbol{\Theta}_{FFNN})),$$

where softmax() is the softmax function, $\mathbf{p}$ is the probability distribution over the relations in the set $\mathcal{R}$, and $\mathbf{F}_{sent}, \mathbf{F}_s, \mathbf{F}_o$ represent the vector representations of the sentence, the subject entity and the object entity, respectively. To capture the context information of the sentence while also capturing the syntactic structure of the sentence, a very similar procedure as (Marcheggiani and Titov, 2017) (i.e. the model we introduced in Section 10.2 for SRL) is adopted to learn the word representations, which are then utilized to learn the representations for the sentence, subject entity and object entity. The major difference is that a self-loop is introduced to include the word itself during representation updating in Eq. (10.1). In other words, $\mathcal{N}(v_i)$ in Eq. (10.1) for RE consists of the node v_i, and its in-going and out-going neighbors. They also empirically find that including the direction and edge label information does not help for the RE task.

Given the word representations from the model consisting of L graph filtering layers described above, the representations for sentence, the subject entity and object entity are obtained by max pooling as:

$$\begin{aligned}
\mathbf{F}_{sent} &= \max(\mathbf{F}^{(L)}),\\
\mathbf{h}_s &= \max(\mathbf{F}^{(L)}[s1 : s2]),\\
\mathbf{h}_o &= \max(\mathbf{F}^{(L)}[o1 : o2]),
\end{aligned} \tag{10.2}$$

where $\mathbf{F}^{(L)}$, $\mathbf{F}^{(L)}[s1 : s2]$, and $\mathbf{F}^{(L)}[o1 : o2]$ denote the sequence of word representations for the entire sentence, the subject entity and the object entity, respectively. The max-pooling operation takes the maximum of each dimension


The Hanging Gardens, in [Mumbai], also known as Pherozeshah Mehta Gardens, are terraced gardens … They provide sunset views over the [Arabian Sea] …

Mumbai (also known as Bombay, the official name until 1995) is the capital city of the Indian state of Maharashtra. It is the most populous city in India …

The Arabian Sea is a region of the northern Indian Ocean bounded on the north by Pakistan and Iran, on the west by northeastern Somalia and the Arabian Peninsula, and on the east by India …

Q: {Hanging gardens of Mumbai, country, ?}

Options: {Iran, India, Pakistan, Somalia, …}

Figure 10.3 A sample from the WIKIHOP dataset

and thus results in a vector with the same dimension as the word representation.

10.5 Question Answering

Machine reading comprehension (RC) or question answering (QA) aims to generate the correct answer for a given query/question by consuming and comprehending documents. It is an important but challenging task in NLP. graph neural networks have been widely adopted to enhance the performance of the QA task, especially multi-hop QA (De Cao et al., 2019; Cao et al., 2019; Song et al., 2018a; Tu et al., 2019), where across-document reasoning is needed to answer a given query. This section introduces the multi-hop QA and one of the representative works, which utilize graph neural networks for this task. We first introduce the setting of multi-hop QA based on the WIKIHOP dataset (Welbl et al., 2018), which is created specifically for evaluating multi-hop QA models. We then describe the entity-GCN proposed in (De Cao et al., 2019) to tackle the multi-hop QA task.

10.5.1 The Multi-hop QA Task

In this subsection, we briefly discuss the setting of multi-hop QA based on the WIKIHOP dataset. The WIKIHOP dataset consists of a set of QA samples. Each sample can be denoted as a tuple $(q, S_q, C_q, a^\star)$, where q is a query/question, S_q is a set of supporting documents, C_q is a set of candidate answers to be chosen from (all of which are entities in the set of supporting documents S_q) and $a^\star \in C_q$ is the correct answer to the query. Instead of natural language, the query q is given in the form of a tuple $(s, r, ?)$, where s is the subject, r denotes the relation, and the object entity is unknown (marked as "?") to be inferred from the support documents. A sample from the WIKIHOP dataset is shown in Figure 10.3, where the goal is to choose the correct "country" for the *Hanging Gardens of Mumbai* from the candidate set C_q ={Iran, India, Pakistan, Somalia}. In this example, to find the correct answer for the query, multi-hop reasoning is required: 1) from the first document, it can be figured out that *Hanging Gardens* are located in *Mumbai*; and 2) then, from the second document, it can be found that *Mumbai* is a city in *India*, which, together with the first evidence, can lead to the correct answer for the query. The goal of multi-hop QA is to learn a model that can identify the correct answer $a^\star$ for a given query q from the candidate set C_q by consuming and comprehending the set of the support documents S_q.

10.5.2 Entity-GCN

To capture the relations between the entities within- and across-documents and consequently help the reasoning process across documents, each sample $(q, S_q, C_q, a^\star)$ of the multi-hop QA task is organized into a graph by connecting mentions of candidate answers within and across the supporting documents. A generalized graph neural network model (i.e., Entity-GCN) is then proposed to learn the node representations, which are later used to identify the correct answer from the candidate sets for the given query. Note that L graph filtering layers are applied to ensure that each mention (or node) can access rich information from a wide range of neighborhoods. Next, we first describe how the graph is built and then introduce the process of solving the QA task using the proposed Entity-GCN.

Entity Graph

For a given sample $(q, S_q, C_q, a^\star)$, to build a graph, the mentions of entities in $C_q \cup \{s\}$ are identified from the supporting document set S_q, and each mention is considered as a node in the graph. These mentions include 1) entities in S_q

that exactly match an element in $C_q \cup \{s\}$ and 2) entities that are in the same co-reference chain as an element in $C_q \cup \{s\}$. An end-to-end co-reference resolution technique (Lee et al., 2017) is utilized to discover the co-reference chains. Various types of edges are constructed to connect these mentions (or nodes) as follows: 1) "Match": two mentions (either within or across documents) are connected by a "Match" edge, if they are identical; 2) "DOC-BASED": two mentions are connected via "DOC-BASED" if they co-occur in the same support document; and 3) "COREF": two mentions are connected by a "COREF" edge if they are in the same co-reference chain. These three types of edges describe three different types of relations between these mentions. Besides, to avoid disconnected components in the graph, the fourth type of edges is added between any pairs of nodes that are not connected. These edges are denoted as "COMPLEMENT" edges, which make the graph a complete graph.

Multi-step Reasoning with Entity-GCN on Entity Graph

To approach multi-step reasoning, a generalized graph neural network model is proposed to transform and propagate the node representations through the built entity graph. Specifically, the graph filter (for the l-th layer) in Entity-GCN can be regarded as instantiating the MPNN framework in Eq. (5.40) to deal with edges of different types as:

$$\mathbf{m}_i^{(l-1)} = \mathbf{F}_i^{(l-1)}\mathbf{\Theta}_s^{(l-1)} + \frac{1}{|\mathcal{N}(v_i)|}\sum_{r\in\mathcal{R}}\sum_{v_j\in\mathcal{N}_r(v_i)} \mathbf{F}_j^{(l-1)}\mathbf{\Theta}_r^{(l-1)}, \tag{10.3}$$

$$\mathbf{a}_i^{(l-1)} = \sigma\left(\left[\mathbf{m}_i^{(l)}, \mathbf{F}_i^{(l-1)}\right]\mathbf{\Theta}_a^{(l-1)}\right), \tag{10.4}$$

$$\mathbf{h}_i^{(l)} = \rho\left(\mathbf{m}_i^{(l-1)}\right) \odot \mathbf{a}_i^{(l-1)} + \mathbf{F}_i^{(l-1)} \odot \left(1 - \mathbf{a}_i^{(l-1)}\right), \tag{10.5}$$

where $\mathcal{R} = \{MATCH, DOC\text{-}BASED, COREF, COMPLEMENT\}$ denotes the set of types of edges, $\mathcal{N}_r(v_i)$ is the set of nodes connected with node v_i through edges of the type r, $\mathbf{\Theta}_r^{(l-1)}$ indicates parameters shared by edges of the type r and $\mathbf{\Theta}_s^{(l-1)}$ and $\mathbf{\Theta}_a^{(l-1)}$ are shared by all nodes. The output in Eq. (10.4) is served as a gating system to control the information flow in the message update part of Eq. (10.5). The representation for each node v_i is initialized as:

$$\mathbf{F}_i^{(0)} = f_x(\mathbf{q}, \mathbf{x}_i),$$

where $\mathbf{q}$ denotes the query representation from the pre-trained model ELMo (Peters et al., 2018) and $\mathbf{x}_i$ is the pre-trained representation for node v_i from ELMo and $f_x(,)$ is parameterized by a feed-forward neural network.

The final node representations $\mathbf{F}_i^{(L)}$ from the Entity-GCN with L graph filtering layers are used to select the answer for the given query from the candidate set. In detail, the probability of selecting a candidate $c \in C_q$ as the answer is modeled as:

$$P\left(c|q, C_q, S_q\right) \propto \exp\left(\max_{v_i \in \mathcal{M}_c} f_o\left(\left[\mathbf{q}, \mathbf{F}_i^{(L)}\right]\right)\right),$$

where f_o is a parameterized transformation, $\mathcal{M}_c$ is the set of mentions corresponding to the candidate c, and the max operator is to select the mention in $\mathcal{M}_c$ with the largest predicted probability for the candidate. In (Song et al., 2018a), instead of selecting the mention with the largest probability in $\mathcal{M}_c$, all mentions of a candidate c are utilized to model $P\left(c|q, C_q, S_q\right)$. In particular,

$$P\left(c|q, C_q, S_q\right) = \frac{\sum\limits_{v_i \in \mathcal{M}_c} \alpha_i}{\sum\limits_{v_i \in \mathcal{M}} \alpha_i},$$

where we use $\mathcal{M}$ to denote all the mentions, i.e., all nodes in the entity graph and α_i is modeled by the softmax function as:

$$\alpha_i = \frac{\exp\left(f_o\left(\left[\mathbf{q}, \mathbf{F}_i^{(L)}\right]\right)\right)}{\sum\limits_{v_i \in \mathcal{M}} \exp\left(f_o\left(\left[\mathbf{q}, \mathbf{F}_i^{(L)}\right]\right)\right)}.$$

10.6 Graph to Sequence Learning

Sequence to sequence models have been broadly applied to natural language processing tasks such as neural machine translation (NMT) (Bahdanau et al., 2014) and natural language generation (NLG) (Song et al., 2017). Most of these proposed models can be viewed as encoder-decoder models. In a encoder-decoder model, an encoder takes a sequence of tokens as input and encodes it into a sequence of continuous vector representations. Then, a decoder takes the encoded vector representations as input and outputs a new target sequence. Usually, recurrent neural networks (RNNs) and its variants serve as both the encoder and the decoder. As the natural languages can be represented in terms of graphs, graph to sequence models have emerged to tackle various tasks in NLP, such as neural machine translation (NMT) (Marcheggiani et al., 2018; Beck et al., 2018) (see Section 10.3 for details) and AMR-to-text (Cohen, 2019; Song et al., 2018b). These graph to sequence models usually utilize graph neural networks as the encoder (or a part of the encoder) while still adopting RNN and its variants as its decoder. Specifically, the graph neural network model described in Eq. (10.1) (Marcheggiani and Titov, 2017) is utilized as encoder in (Marcheggiani et al., 2018; Song et al., 2018b; Cohen, 2019) for neural machine translation and AMR-to-text tasks. A general encoder-decoder graph2seq framework for graph to sequence learning is proposed in (Xu et al., 2018b). It utilizes the graph neural network model as the encoder, and an attention mechanism equipped RNN model as the decoder. We first describe the GNN-based encoder model and then briefly describe the decoder.

GNN-based Encoder

Most of graphs in NLP applications such as the AMR and syntactic dependency trees are directed. Hence, the GNN-based encoder in graph2seq is designed to differentiate the incoming and outgoing neighbors while aggregating information. Specially, for a node v_i, its neighbors are split into two sets – the incoming neighbors $\mathcal{N}_{\text{in}}(v_i)$ and the outgoing neighbors $\mathcal{N}_{\text{out}}(v_i)$. The aggregation operation in GraphSAGE-Filter (See details on GraphSAGE-Filter in Section 5.3.2) is used to aggregate and update the node representations. Specifically, two node representations for each node are maintained, i.e., the in-representation and the out-representation. The updating process for node v_i in the l-th layer can be expressed as:

$$\begin{aligned}
\mathbf{F}^{(l)}_{\mathcal{N}_{\text{in}}(v_i)} &= \text{AGGREGATE}(\{\mathbf{F}^{(l-1)}_{\text{out}}(v_j), \forall v_j \in \mathcal{N}_{\text{in}}(v_i)\}),\\
\mathbf{F}^{(l)}_{\text{in}}(v_i) &= \sigma\left([\mathbf{F}^{(l-1)}_{\text{in}}(v_i), \mathbf{F}^{(l)}_{\text{in}}(\mathcal{N}_{(v_i)})]\mathbf{\Theta}^{(l-1)}_{\text{in}}\right),\\
\mathbf{F}^{(l)}_{\mathcal{N}_{\text{out}}(v_i)} &= \text{AGGREGATE}(\{\mathbf{F}^{(l-1)}_{\text{in}}(v_j), \forall v_j \in \mathcal{N}_{\text{out}}(v_i)\}),\\
\mathbf{F}^{(l)}_{\text{out}}(v_i) &= \sigma\left([\mathbf{F}^{(l-1)}_{\text{out}}(v_i), \mathbf{F}^{(l)}_{\mathcal{N}_{\text{out}}(v_i)}]\mathbf{\Theta}^{(l-1)}_{\text{out}}\right),
\end{aligned}$$

where $\mathbf{F}^{(l)}_{\text{in}}(v_i)$ and $\mathbf{F}^{(l)}_{\text{out}}(v_i)$ denote the in- and out-representations for node v_i after l-th layer. As introduced for the GraphSAGE-Filter in Section 5.3.2, various designs for AGGREGATE() functions can be adopted. The final in- and out-representations after L graph filtering layers are denoted as $\mathbf{F}^{(L)}_{\text{in}}(v_i)$ and $\mathbf{F}^{(L)}_{\text{out}}(v_i)$, respectively. These two types of representations are concatenated to generate the final representations containing information from both directions as:

$$\mathbf{F}^{(L)}(v_i) = \left[\mathbf{F}^{(L)}_{\text{in}}(v_i), \mathbf{F}^{(L)}_{\text{out}}(v_i)\right].$$

After obtained the node representations, a graph representation is also generated by using pooling methods, which is used to initialize the decoder. The pooling process can be expressed as:

$$\mathbf{F}_G = \text{Pool}\left(\left\{\mathbf{F}^{(L}(v_i), \forall v_i \in \mathcal{V}\right\}\right).$$

Here, various flat pooling methods such as max pooling and average pooling can be adopted. The decoder is modeled by an attention-based recurrent neural network. It attends to all node representations when generating each token of the sequence. Note that the graph representation $\mathbf{F}_G$ is utilized as the initial state of the RNN decoder.

10.7 Graph Neural Networks on Knowledge Graphs

Formally, a knowledge graph $G = (\mathcal{V}, \mathcal{E}, \mathcal{R})$ consists a set of nodes $\mathcal{V}$, a set of relational edges $\mathcal{E}$ and a set of relations $\mathcal{R}$. The nodes are various types of entities and attributes, while the edges include different types of relations between the nodes. Specifically, an edge $e \in \mathcal{E}$ can be represented as a triplet (s, r, t) where $s, t \in \mathcal{V}$ are the source and target nodes of the edge respectively, and $r \in \mathcal{R}$ denotes the relation between them. Graph neural networks have been extended to knowledge graphs to learn node representations and thus facilitate various downstream tasks, including knowledge graph completion (Hamaguchi et al., 2017; Schlichtkrull et al., 2018; Nathani et al., 2019; Shang et al., 2019a; Wang et al., 2019f), node importance estimation (Park et al., 2019), entity linking (Zhang et al., 2019b) and cross-language knowledge graph alignment (Wang et al., 2018c; Xu et al., 2019e). The major difference between the knowledge graphs and simple graphs is the relational information, which is important to consider when designing graph neural networks for knowledge graphs. In this section, we first describe how graph neural networks are generalized to knowledge graphs. Especially, there are majorly two ways to deal with the relational edges in knowledge graphs: 1) incorporating the relational information of the edges into the design of graph filters; and 2) transforming the relational knowledge graph into a simple undirected graph by capturing the relational information. Then, we use the task of knowledge graph completion as an example to illustrate GNN based applications on knowledge graphs.

10.7.1 Graph Filters for Knowledge Graphs

Various graph filters have been specifically designed for knowledge graphs. We describe representative ones next. The GGNN-Filter described in Eq. (5.22) is adapted to knowledge graphs (Schlichtkrull et al., 2018) as:

$$\mathbf{F}_i^{(l)} = \sum_{r \in \mathcal{R}} \sum_{v_j \in \mathcal{N}_r(v_i)} \frac{1}{|\mathcal{N}_r(v_i)|} \mathbf{F}_j^{(l-1)} \mathbf{\Theta}_r^{(l-1)} + \mathbf{F}_i^{(l-1)} \mathbf{\Theta}_0^{(l-1)}, \tag{10.6}$$

where $\mathcal{N}_r(v_i)$ denotes the set of neighbors that connect to node v_i through the relation r. It can be defined as:

$$\mathcal{N}(v_i) = \{v_j | (v_j, r, v_i) \in \mathcal{E}\}.$$

In Eq. (10.6), the parameters $\mathbf{\Theta}_r^{(l-1)}$ are shared by the edges with the same relation $r \in \mathcal{R}$. Similar ideas can be also found in (Hamaguchi et al., 2017). Note that the Entity-GCN described in Section 10.5.2 is inspired by the graph filter in Eq. (10.6). In (Shang et al., 2019a), instead of learning different transformation parameters for different relations, a scalar score is learned to capture the importance for each relation. It leads to the following graph filtering operation:

$$\mathbf{F}_i^{(l)} = \sum_{r \in \mathcal{R}} \sum_{v_j \in \mathcal{N}_r(v_i)} \frac{1}{|\mathcal{N}_r(v_i)|} \alpha_r^{(l)} \mathbf{F}_j^{(l-1)} \mathbf{\Theta}^{(l-1)} + \mathbf{F}_i^{(l-1)} \mathbf{\Theta}_0^{(l-1)}, \tag{10.7}$$

where $\alpha_r^{(l)}$ is the importance score to be learned for the relation r.

To reduce the parameters involved in Eq. (10.6), relation embeddings are learned for different relations in (Vashishth et al., 2019). Specifically, the relation embeddings for all relations in $\mathcal{R}$ after $l-1$ layer can be denoted as $\mathbf{Z}^{(l-1)}$ with $\mathbf{Z}_r^{(l-1)}$ the embedding for relation r. The relation embeddings can be updated for the l-th layer as:

$$\mathbf{Z}^{(l)} = \mathbf{Z}^{(l-1)} \mathbf{\Theta}_{rel}^{(l-1)},$$

where $\mathbf{\Theta}_{rel}^{(l)}$ are the parameters to be learned. We use $\mathcal{N}(v_i)$ to denote the set of neighbors of node v_i, which contains nodes that connect to v_i with different relations. Hence, we use (v_j, r) to indicate a neighbor of v_i in $\mathcal{N}(v_i)$, where v_j is the node connecting with v_i through the relation r. Furthermore, in (Vashishth et al., 2019), the reverse edge of any edge in $\mathcal{E}$ is also treated as an edge. In other words, if $(v_i, r, v_j) \in \mathcal{E}$, $(v_j, \hat{r}, v_i)$ is also considered as an edge with $\hat{r}$ as the reverse relation of r. Note that, for convenience, we abuse the notation $\mathcal{E}$ and $\mathcal{R}$ a little bit to denote the augmented edge set and relation set. The relations now have directions and we use $dir(r)$ to denote the direction of a relation r. Specifically, $dir(r) = 1$ for all the original relations, while $dir(\hat{r}) = -1$ for all the reverse relations. The filtering operation is then designed as:

$$\mathbf{F}_i^{(l)} = \sum_{(v_j, r) \in \mathcal{N}(v_i)} \phi(\mathbf{F}_j^{(l-1)}, \mathbf{Z}_r^{l-1}) \mathbf{\Theta}_{dir(r)}^{(l-1)}, \tag{10.8}$$

where $\phi(,)$ denotes non-parameterized operations such as subtraction and multiplication and $\mathbf{\Theta}_{dir(r)}^{(l-1)}$ are parameters shared by all the relations with the same direction.

10.7.2 Transforming Knowledge Graphs to Simple Graphs

In (Wang et al., 2018c), instead of designing specific graph filtering operations for knowledge graphs, a simple graph is built to capture the directed relational information in knowledge graphs. Then, existing graph filtering operations can be naturally applied to the transformed simple graph.

Two scores are proposed to measure the influence of an entity to another entity through a specific type of relation r as:

$$\text{fun}(r) = \frac{\text{\#Source\_with\_r}}{\text{\#Edges\_with\_r}},$$
$$\text{ifun}(r) = \frac{\text{\#Taget\_with\_r}}{\text{\#Edges\_with\_r}},$$

where #Edges_with_r is the total number of edges with the relation r, #Source_with_r denotes the number of unique source entities with relation r and #Target_with_r indicates the number of unique target entities with relation r. Then, the overall influence of the entity v_i to the entity v_j is defined as:

$$\mathbf{A}[i,j] = \sum_{(v_i,r,v_j)\in\mathcal{E}} \text{ifun}(r) + \sum_{(v_j,r,v_i)\in\mathcal{E}} \text{fun}(r),$$

where $\mathbf{A}[i,j]$ is the i,j-th element for the adjacency matrix $\mathbf{A}$ of the generated simple graph.

10.7.3 Knowledge Graph Completion

Knowledge graph completion, which aims to predict the relation between a pair of disconnected entities, is an important task as knowledge graphs are usually incomplete or fast evolving with new entities emerging. Specifically, the task is to predict whether a given triplet (s,r,t) is a real relation or not. To achieve this goal, we need to assign a score $f(s,r,t)$ to the triplet (s,r,t) to measure the probability of the triplet being a real relation. Especially, the DistMult factorization (Yang et al., 2014) is adopted as the scoring function, which can be expressed as:

$$f(s,r,t) = \mathbf{F}_s^{(L)\top}\mathbf{R}_r\mathbf{F}_t^{(L)},$$

where $\mathbf{F}_s^{(L)}$ and $\mathbf{F}_t^{(L)}$ are the representations of source node s and target node t, respectively. They are learned by graph neural networks after L filtering layers; $\mathbf{R}_r$ is a diagonal matrix corresponding to the relation r to be learned during training. The model can be trained using negative sampling with cross-entropy loss. In particular, for each observed edge sample $e \in \mathcal{E}$, k negative samples are generated by randomly replacing either its subject or object with another entity. With the observed samples and the negative samples, the cross-entropy loss to be optimized can be expressed as:

$$\mathcal{L} = -\frac{1}{(1+k)|\mathcal{E}|} \sum_{(s,r,o,y)\in\mathcal{T}} y \log \sigma\left(f(s,r,o)\right) + (1-y)\log\left(1-\sigma\left(f(s,r,o)\right)\right),$$

where $\mathcal{T}$ denotes the set of the positive samples observed in $\mathcal{E}$ and randomly generated negative samples and y is an indicator that is set to 1 for the observed samples and 0 for the negative samples.

10.8 Conclusion

In this chapter, we introduce how graph neural networks can be applied to natural language processing. We present representative tasks in natural language processing, including semantic role labelling, relation extraction, question answering, and graph to sequence learning, and describe how graph neural networks can be employed to advance their corresponding models' performance. We also discuss knowledge graphs, which are widely used in many NLP tasks and present how graph neural networks can be generalized to knowledge graphs.

10.9 Further Reading

Besides graph neural networks, the Graph-LSTM algorithms we introduced in Section 9.3 have also been adopted to advance the relation extraction tasks (Miwa and Bansal, 2016; Song et al., 2018c). In addition, graph neural networks have been applied to many other NLP tasks such as abusive language detection (Mishra et al., 2019), neural summarization (Fernandes et al., 2018), and text classification (Yao et al., 2019). The Transformer (Vaswani et al., 2017) has been widely adopted to deal with sequences in natural language processing. The pre-trained model BERT (Devlin et al., 2018), which is built upon transformer, has advanced many tasks in NLP. When applying to a given sequence, the transformer can be regarded as a special graph neural network. It is applied to the graph induced from the input sequence. In detail, the sequence can be regarded as a fully connected graph, where elements in the sequence are treated as the nodes. Then a single self-attention layer in the transformer is equivalent to the GAT-Filter layer (see Section 5.3.2 for details of GAT-Filter).