國立台灣大學電機資訊學院資訊工程學系

碩士論文

Department of Computer Science and Information Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master Thesis

利用語法結構之雙向遞迴類神經網路

於命名實體辨識之研究

Leveraging Linguistic Structures for Named Entity Recognition with Bidirectional Recursive Neural Networks

李朋軒

Peng-Hsuan Li

指導教授： 許永真 博士

馬偉雲 博士

Advisor: Jane Yung-jen Hsu, Ph.D.

Wei-Yun Ma, Ph.D.

中華民國 106 年 11 月

November, 2017

doi:10.6342/NTU201704465

doi:10.6342/NTU201704465

Acknowledgments

感謝許永真老師指導我研究。大學部時，許永真老師和王詩翰學長帶領我探索各個研究問題與技術，引領我完成一個研究專題。碩士班時，許永真老師給予我各個研究方向的回饋，也給我許多碩士論文撰寫建議。

感謝馬偉雲老師指導我研究。碩士班時，馬偉雲老師帶領我探索一個研究領域，指導我在一個具體的研究題目上得到成果，並教導我投稿和撰寫會議論文。至今，馬偉雲老師持續給我進一步研究方向建議和回饋。

感謝實驗室的同學們合作討論，感謝我的家人，感謝許許多多的人在我成長過程中帶來正面影響。

doi:10.6342/NTU201704465

doi:10.6342/NTU201704465

Abstract

Named Entity Recognition (NER) is an important task which locates proper names in text for downstream tasks, e.g. to facilitate natural language understanding. The problem is often casted from structured prediction of text chunks to sequential labeling of tokens. Such sequential approaches have achieved high performance with models like conditional random fields and recurrent neural networks. However, named entities should be linguistic constituents, and sequential token labeling neglects this information.

In the thesis, we propose a constituency-oriented approach which fully utilizes linguistic structures in text. First, to leverage the prior knowledge of hierarchical phrase structures, we generate parses and alter them into constituency graphs that minimize inconsistencies between parses and named entities. Then, we use Bidirectional Recursive Neural Networks (BRNN) to propagate relevant structure information to each constituent. We use a bottom-up pass to capture the local information and a top-down pass to capture the global information. Experiments show that this approach is comparable to sequential token labeling, and significant improvements can be seen on OntoNotes 5.0 NER, with F1 scores over 87%.

doi:10.6342/NTU201704465

摘要

命名實體辨識(NER)是一個找出文字中的命名實體的重要任務，其產出能提供給下游的任務比如自然語言理解使用。此問題常從命名實體所在的文字區段的預測被轉型爲線性地預測每個單詞是否屬於某一命名實體的一部分。利用CRF與RNN等模型，這類轉型後的方法取得了很好的成果。然而，每個命名實體都應該是一個語法單元，而線性單詞預測的方法忽略這個資訊。

在本論文中，我們提出一個語法導向的方法以完整利用文字裡的語言結構。要利用階層性的詞組結構，我們首先產生語法剖析樹並將之改變爲最小化剖析樹與命名實體之間的不一致的語法圖。然後我們利用雙向遞迴類神經網路(BRNN)去傳遞相關的結構資訊到每一個語法單元。我們利用一個由下往上的遍歷來蒐集局部資訊，以及一個由上往下的遍歷來蒐集全域資訊。實驗顯示此方法可和線性單詞標記法相比，並在OntoNotes 5.0 NER語料上取得了超過87% F1分數的顯著進步。

doi:10.6342/NTU201704465

Contents

doi:10.6342/NTU201704465

doi:10.6342/NTU201704465

doi:10.6342/NTU201704465

doi:10.6342/NTU201704465

List of Figures

doi:10.6342/NTU201704465

doi:10.6342/NTU201704465

List of Tables

doi:10.6342/NTU201704465

doi:10.6342/NTU201704465

Chapter 1

Introduction

In this chapter, we start off introducing the background of the thesis. Then our motivation and goals are given. Finally, the following chapters in the thesis are briefly summarized.

1.1 Background

Named Entities (NEs) are text chunks that represent names, and they are sometimes simply referred to as *names* or *entities*. The types of names that are often wanted to be recognized include *PERSON*, *ORGANIZATION*, and *LOCATION*. While in specific domains such as biomedicine, each molecule can be seen as a category of named entities, many other categories of general named entities have also been proposed, e.g. *WORK OF ART* and *LAW*.

Named Entity Recognition (NER), which can be seen as a combined task of lo-

doi:10.6342/NTU201704465

cating and classifying named entities, is an important task of information extraction systems. Recent important benchmark datasets of the general domain include the dataset of CoNLL 2003 shared task [31] and the dataset of the OntoNotes project [13]. CoNLL 2003 is the Reuters corpus with NER annotations, and OntoNotes 5.0 boasts multilevel annotations, e.g. TreeBank, PropBank, and NE, for diverse sources of texts.

NER problems are often casted from structured prediction of text chunks to sequential labeling of tokens (Figure 1.1). This is done by labeling each token as a part of a named entity chunk, e.g. "Begin_Person". Such approaches achieve high performances in the benchmark datasets [25, 22, 3].

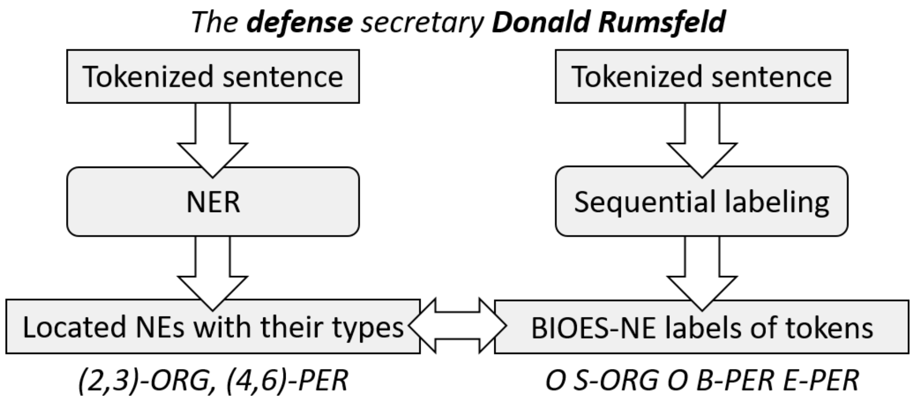


Figure 1.1: The NER task and a sequential labeling approach with BIOES sequential labels (**B**egin, **I**nside, **O**utside, **E**nd, **S**ingle).

Being formulated as a sequential labeling problem, NER systems could be implemented by models which compute hidden states for each token. These hidden state features are then used to predict the sequential label of a token. Such kind of

doi:10.6342/NTU201704465

models include conditional random fields and recurrent neural networks. With both forward and backward directions, bidirectional networks learn how to propagate the information of a token sequence to each token. Bi-LSTM-CNN, a variant of such models, is shown to accomplish state-of-the-art results on both CoNLL 2003 and OntoNotes 5.0 NER [3].

1.2 Motivation

According to analyses, most named entity chunks are actually linguistic constituents, e.g. noun phrases, and additional linguistic information other than word orders should be intuitively useful (Figure 1.2). However, due to the hierarchical chunking nature of phrase structures, it is intrinsically hard for sequential labeling token-based NER models to take advantage of them. Unfortunately for constituent-based NER models, the inconsistencies between constituency parses and named entities pose another challenge: the recall of such models is capped by the proportion of named entities that correspond to some constituents

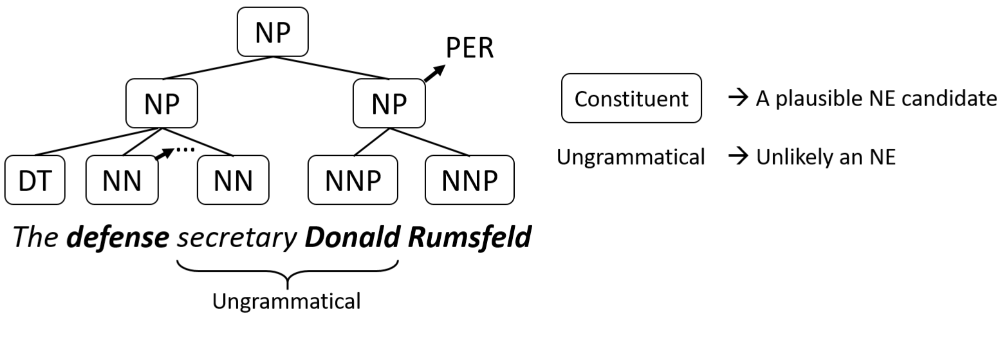


Figure 1.2: Linguistic structures and NER.

doi:10.6342/NTU201704465

Provided that both parse trees and NEs are given, they can be made consistent by flattening the trees and then adding new nodes [10]. However, this condition is not practical for NER systems that are dependent on parses. Instead, for any NER training corpora with or without constituency parse annotations, readily available parsers can be used. It is then desirable to have algorithms that can still alter those parser-generated parses to make them more consistent without actually knowing NE locations.

Additionally, as the approach shifts from sequential labeling token-based NER to tree-structured constituent-based NER, *recursive* neural networks should be considered. Recurrent neural networks are shown to be powerful on sequential labeling NER, and recursive networks are the generalization that can operate on tree structures. To capture the relevant information for each token, bidirectional recurrent networks have two passes for left and right context respectively. For recursive networks, they could have a bottom-up pass to capture local information and a top-down pass to capture global information.

1.3 Objective

To leverage linguistic structures in texts for NER, we want to

- Mitigate the inconsistencies between parsing and NER by restructuring algorithms, and
- Utilize prior linguistic structure information with constituent-based Bidirectional Recursive Neural Networks (BRNN).

doi:10.6342/NTU201704465

1.4 Outline of the Thesis

Chapter 2 explores previous work on NER, treatments for the consistency problem, and models closely related to the proposed BRNN-CNN.

Chapter 3 first states the NER problem to solve, and then shows the overview of the system proposed by this thesis. Chapter 4 elaborates the first functional block of the system: constituency graph generation. This chapter covers the construction of linguistic structures, inconsistencies between the structures and NER, and algorithms that mitigate these inconsistencies. Finally, after linguistic structures needed are defined, Chapter 5 formulates the proposed model and the features used.

In Chapter 6, evaluation setup about tuning, training, and testing of the system on different datasets are documented. Experiment results and analyses of different aspects of the approach are given.

Chapter 7 summarizes the contribution of the thesis as well as possible future research directions.

doi:10.6342/NTU201704465

Chapter 2

Related Work

In this chapter, related researches of the NER problem and neural models are presented first. The last two sections then introduce two approaches that are most related to the thesis.

2.1 NER

Studies of named entity recognition can be dated back to the Message Understanding Conference-6 at 1995 [12]. NER systems on this well-studied MUC-7 dataset [2] have achieved near-human performances (93% against 97%) [21]. However, NER remains an active and challenging research topic to date with various complications. These include more classes of general named entities, more fine-grained categories, an indefinite number of domain-specific types, diverse sources of corpora, crowd-based external knowledge, and joint tasks of related problems

doi:10.6342/NTU201704465

[31, 13, 17, 7, 9].

The Conference on Computational Language Learning (CoNLL) organized by SIGNLL includes a shared Natural Language Processing (NLP) task every year. In 2003, CoNLL held a language-independent NER shared task [31]. Since then, its English corpus, the Reuters Corpus Volume 1 (RCV1) [19] annotated with sequential NE labels, has become an widely used benchmark for recent systems.

OntoNotes, a project which creates multilingual, multi-source, and substantially larger corpora with multilevel annotations, are first described in 2006 [13]. Particularly, the multilevel annotations make it possible for systems that tackle different tasks in the NER pipeline to share with, compare with, or depend on one another. In addition, joint models that try to solve multiple problems at once are made possible. For NER, OntoNotes release 5.0 [32] annotates more categories of names and numerical quantities which have wider coverage and are more fine-grained. Baselines for various tasks as well as a train-validate-test split which later become the standard have been established for this final release [24]. In 2012, the dataset was used for the multilingual coreference shared task held by CoNLL, and has since been gaining popularity as a benchmark for NE-related tasks.

Traditionally, NER is modeled as a token-based sequential labeling problem by breaking each NE chunk into chunk labels. (An example is shown in Figure 1.1.) The most widely used chunk labels include **B**egin, **I**nside, **O**utside, **E**nd, and **S**ingle. In the famous **BIO** chunk labeling, every token that is not **O**utside any chunk is labeled as **I**nside unless it is the **B**eginning of a chunk. However, for the more complicated **BIOES**, or **BILOU** (**B**egin, **I**nside, **L**ast, **O**utside, **U**nit), chunk labeling, the **L**ast

doi:10.6342/NTU201704465

token of a multi-token chunk is labeled as **E**nd, whereas the token of a **U**nit-length chunk is labeled as **S**ingle. Studies that use the latter chunk labeling scheme have been dominating pure NER tasks (as opposed to joint tasks) [25, 22, 3].

When a large corpus with multiple annotations such as OntoNotes are available, constructing models that are guided by multilevel information is then possible. Intuitively, since all the human-labeled linguistic annotations are sound, models that are trained by more than NER labels should not perform worse than pure NER models. However, these additional labels generated by human are costly and could practically only be obtained in training time, so they cannot be used as features but targets to predict. In other words, to utilize multiple annotations, joint models that tackle several tasks at once must be trained. This kind of models are generally hard to train successfully despite intrinsically having advantages against pure NER models. Joint systems for NER that were successful at their time include one that jointly predicts NE while parsing [10], and ones that does named entity typing, linking and even coreference at once [9, 20].

2.2 Related Neural Models

Neural Networks (NN), the collection of functions composed by linear combinations and nonlinearities, are proved to be able to approximate any continuous functions on a close interval [6]. The universality and empirical results of deep neural networks intrigue to construct end-to-end models that use raw sources of information as features for each domain, e.g. pixels for vision. In 2011, the SENNA

doi:10.6342/NTU201704465

system that almost use raw words as features achieved near state-of-the-art performances on various NLP tasks, including part-of-speech (POS) tagging, chunking, NER, and semantic role labeling, at its time [5].

The actual raw features in text are the sequences of characters, which means the usage of a pre-trained word segmentation system already introduces noises and losses of information. However, for synthetic languages that has multiple phonemes per word like English, training a true end-to-end model is intrinsically hard since each character encodes little semantic information. Still, character-level features have recently been shown to be effective in capturing morphological information inside words and combating the word sparsity problem. State-of-the-art NER systems have been achieved for Spanish and Portuguese by using both word and character embeddings [8]. Some models use word segmentation only as boundaries for the computation of character-level word embeddings by convolutional networks. Such kind of recurrent neural network language models outperforms word-level baselines for several languages with rich morphology (Arabic, Czech, French, German, Spanish, Russian) [15].

While recurrent neural networks repeatedly apply their hidden layers to a sequence of inputs, ***recursive*** neural networks repeatedly apply their hidden layers to a Directed-Acyclic Graph (DAG) of inputs. In other words, recursive networks are generalized recurrent networks with relaxed condition on the dependency of inputs (Figure 2.1). When applied to parse trees, the computed hidden states of each node capture the semantic composition of the corresponding constituent. Thus they have been applied to constructing parses, computing sentence embeddings for sen-

doi:10.6342/NTU201704465

timent analysis and paraphrase detection, and computing additional features (in a top-down fashion) for tokens of a sequential model [27, 30, 28, 29, 26, 14].

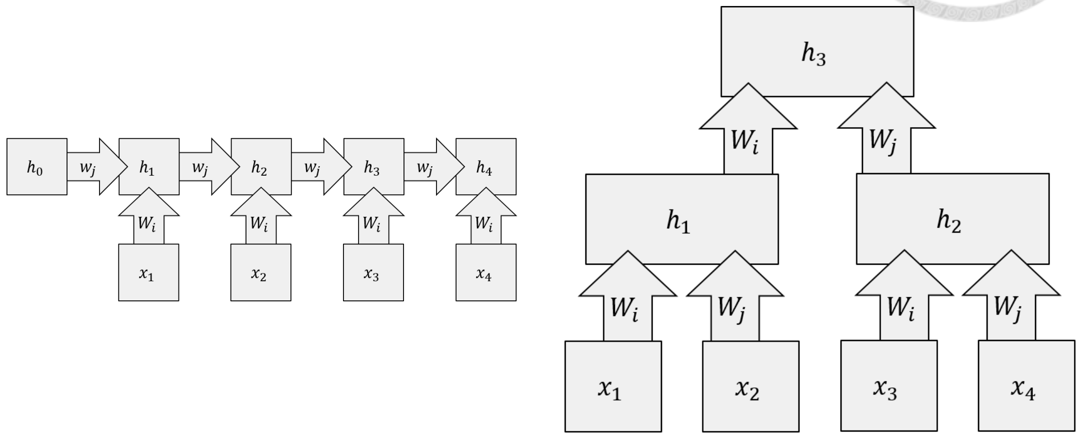


Figure 2.1: Recurrent vs. ***Recursive*** NN.

2.3 Constituents and NER

A complete sentence consists of phrases that are organized in a hierarchical structure. The constituency parse of a sentence is a tree of constituent nodes, where constituents are functional units in a sentence, including words, phrases, clauses, etc. Hence, a named entity should correspond to a constituent, probably a Noun Phrase (NP). A sequential NP-based approach with linear-chain Conditional Random Fields (CRF) has been proposed as part of an ensemble for NER [33]. However, the full potential of constituency structures has not been utilized by the system.

On OntoNotes, both NER and constituency parse annotations are available, so it is proposed to do NER while parsing [10]. However, the parse and NER annotations were found to be inconsistent. NEs might cross constituent boundaries by consisting

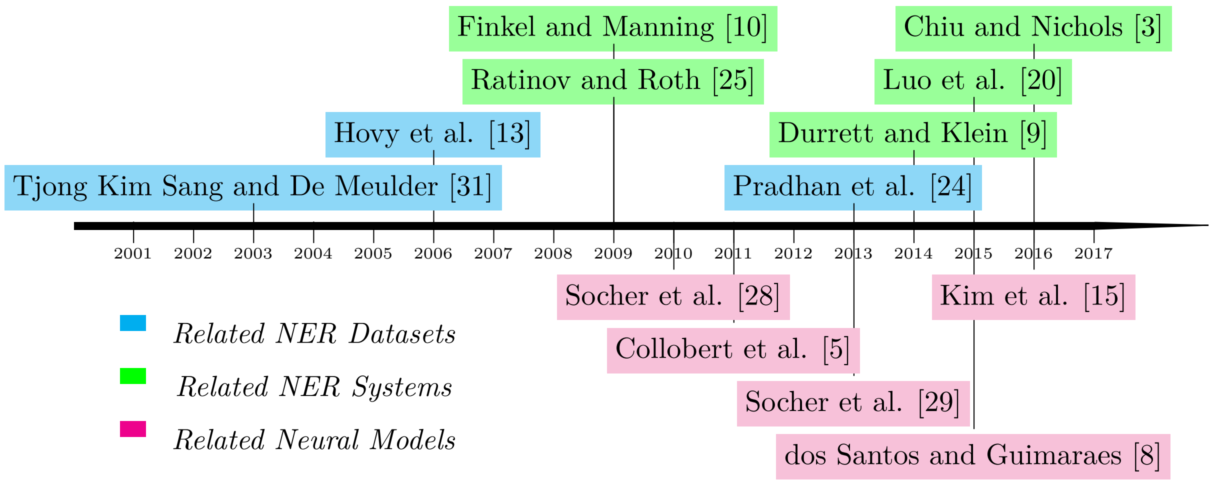
doi:10.6342/NTU201704465

of multiple *sibling* constituents, or even cross tree branches by, for instance, consisting of multiple *cousin* constituents. These inconsistencies were deemed by the authors of that work as annotation errors of the parse trees and were resolved by modifying the dataset. Some subtrees are flattened and smaller constituents are regrouped according to NER annotations. Then the Context Free Grammar (CFG) for parsing was modified so that each nonterminal, e.g. NP, was further lexicalized by adding NE suffixes, e.g. NP-PERSON, NP-LOCATION. A CRF-CFG parser of this grammar was trained on the modified dataset. The method outperformed the same parser of the vanilla grammar on parsing, and surpassed a token-based linear-chain CRF on NER respectively.

2.4 Recurrent NN and NER

A hidden layer of a feed-forward neural network computes a hidden vector from its previous layer, except that the first layer computes from the raw feature vector extracted from a sample. On the other hand, a hidden layer of a recurrent neural network has two input vectors, with the additional one being the output hidden vector from the last time this very layer was applied. Essentially, such kind of networks learn to propagate useful information of previous samples to the current sample, and are suited for classifying sequences of dependent samples. A Long-Short Term Memory (LSTM) is one variation of recurrent neural networks that are oftentimes more successful.

The current state-of-the-art NER system takes a sequential labeling token-based approach with Bi-LSTM-CNNs [3]. In the core of the model, bidirectional LSTM layers learn to propagate the information of the left and the right contexts of a token respectively. The attached CNN learns to compute character-level features to augment other raw features of a token, e.g. the word embedding. Notably, the authors crafted good lexicon features that record if a token is seen in the NE lexicons extracted from SENNA and DBpedia [18].



doi:10.6342/NTU201704465

Chapter 3

Constituency-Oriented NER

The goal of the thesis is to leverage linguistic structures for NER. To this end, we propose a constituency-oriented approach where a constituency graph is generated for each sentence before structure information is utilized to classify each constituent.

3.1 Problem Statement

Let C be the set of named entity categories. Let $S = \{S_i\}$ be the set of tokenized sentences. A sentence S_i is a sequence of tokens $(S_{i1}, S_{i2}, S_{i3}, \ldots, S_{in})$ where n is the number of tokens of S_i. A named entity $e = (i, (j, k), c)$ is the chunk of tokens $(S_{ij}, S_{i(j+1)}, \ldots, S_{i(k-1)})$ with type $c \in C$. The NER problem are thus:

Given C,

Input S,

Output a set of named entities E_a.

doi:10.6342/NTU201704465

The ground truth E_g is unknown to a system in testing time. The quality of the system is determined by the score $F(E_a; E_g)$, where F is an evaluation function.

Abstractly, the problem is to find an NER system which locates and classifies named entities in text. Additionally, the system operates in a per-sentence basis, assuming they are already tokenized. When a user gives such a sentence to the system, chunks of tokens that belong to some predefined named entities categories, as well as which categories they belong to, will be identified. An example is shown in Figure 3.1.

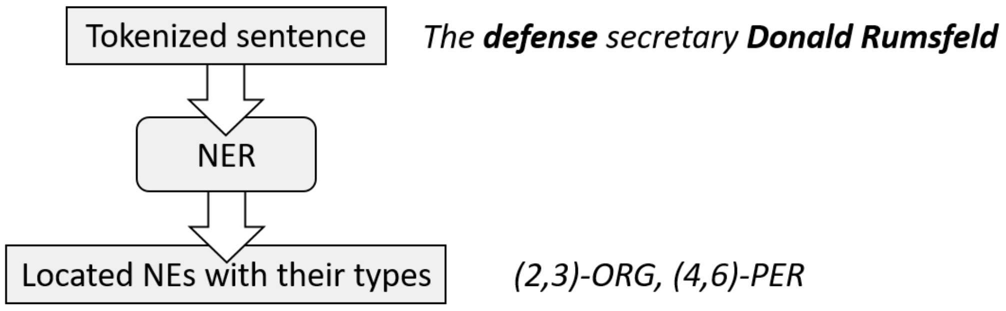

Tokenized sentence
The defense secretary Donald Rumsfeld
NER
Located NEs with their types
(2,3)-ORG, (4,6)-PER

Figure 3.1: An NER system.

3.2 Proposed Solution

In this thesis, a **constituency-oriented approach** for NER is proposed. Figure 3.2 shows the constituency-oriented NER compared to traditional sequential labeling NER. Notably, NER is split into two stages in the proposed approach, of which underlying methods might be swapped independently. In the first stage, a hierarchy of constituents is constructed to take into account additional structural linguistic information. Then in the second stage, constituent-based predictions are made. We suggest that the model underlying the second stage classifies each constituent not only by the constituent itself but also by relevant structures provided by the hierarchy.

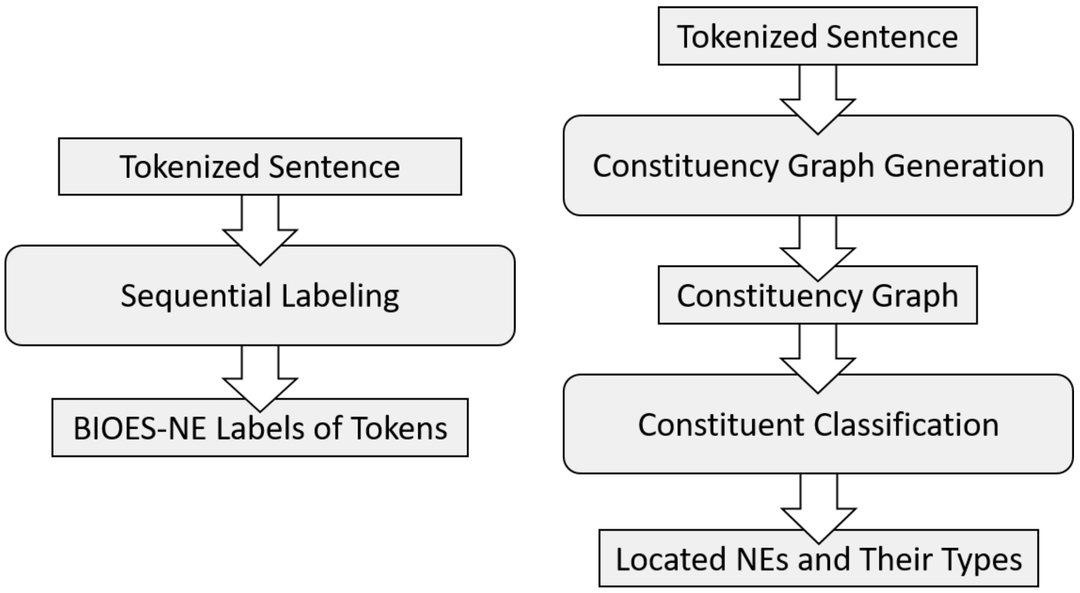

Tokenized Sentence
Sequential Labeling
BIOES-NE Labels of Tokens
Tokenized Sentence
Constituency Graph Generation
Constituency Graph
Constituent Classification
Located NEs and Their Types

Figure 3.2: Comparing classical sequential labeling NER and constituency-oriented NER.

Figure 3.3 shows components proposed for each stage. In constituency graph generation, a constituency parse tree is first constructed but later altered and augmented. These processes make it not a tree anymore, hence the name **constituency graph**. In constituent classification, hidden state features are computed recursively before classification. This is done by **BRNN-CNN**, a specially designed recursive network. Actual functional blocks of the system are briefly introduced in the following, and more details are elaborated in later chapters.

doi:10.6342/NTU201704465

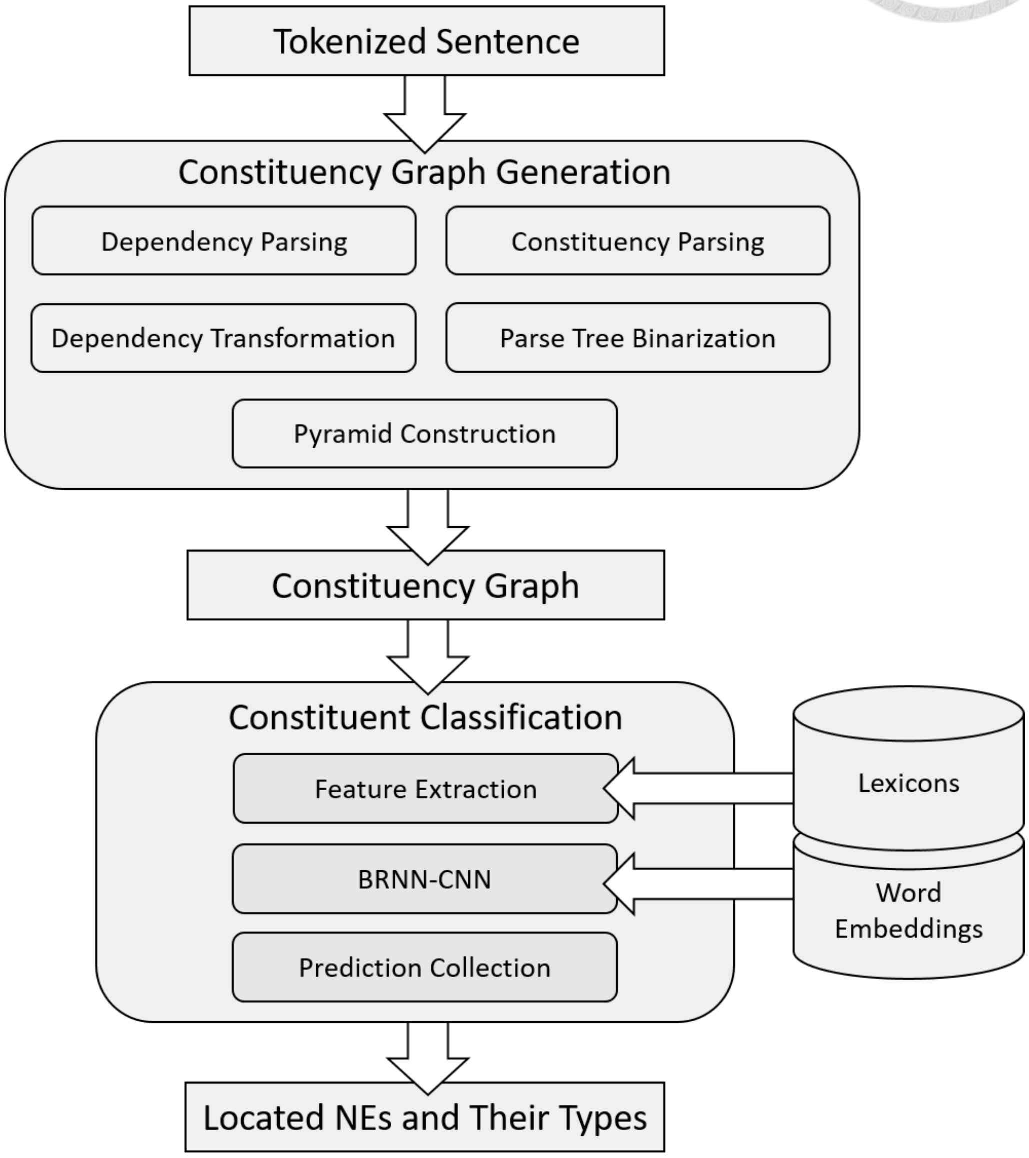


Figure 3.3: High-level overview of the system.

doi:10.6342/NTU201704465

The functional blocks in the first stage constructs a special directed graph, called a constituency graph, from a given tokenized sentence. Several restructuring algorithms are applied to a base parse tree to form the graph. These algorithms increase the consistency between the constituency graph and the (unknown) named entities of a given sentence while preserving linguistic structures hinted by the parse.

The functional blocks in the second stage are tasked with classifying constituents by a constituency graph. To utilize the structural linguistic information, a special recursive network, BRNN-CNN, is proposed. Crucially, two Directed Acyclic Graphs (DAGs) are formed by considering only bottom-up or top-down links in the hierarchical constituency graph. Then BRNN-CNN computes two hidden state features for each node recursively. The bottom-up pass captures the semantic composition of local information for each constituent. The top-down pass captures the global information of the structures containing each constituent. Together, the two hidden state features contain the relevant information for the identification of named entity constituents.

doi:10.6342/NTU201704465

Chapter 4

Constituency Graph Generation

The first stage of the proposed approach regards the construction of a special graph, called a constituency graph, for each given sentence. Initially, a constituency parse tree is constructed, either by direct constituency parsing or by transforming a dependency parse. Then the parse tree is binarized and augmented to form a hierarchical graph. The applied algorithms increase the consistency between the constituency graph and the unknown named entities of a given sentence while preserving linguistic structures hinted by the parse.

4.1 Constituency Graph

A constituency graph of hierarchical nodes is constructed for each given sentence. The graph must meet the following conditions.

- Every node in the graph corresponds to a chunk of tokens in the sentence.

doi:10.6342/NTU201704465

- For every pair of nodes linked by some edges, one of the corresponding chunks contains the other.

Throughout the thesis, some terms are used as the following for simplicity.

- A chunk of tokens in the sentence that corresponds to some nodes is called a *constituent*.
- For every pair of nodes linked by some edges, one is designated to be the *parent* and the other the *child* such that the parent constituent contains the child constituent.
- An edge is said to be a *bottom-up* link if it points to the parent, otherwise a *top-down* link.
- Nodes of which bottom-up links point to the same parent are called *siblings*.
- Two siblings of which constituents are right next to each other are called the *left sibling* or the *right sibling* of each other, depending on which constituent is on the left and which is on the right.

Intuitively, a constituency graph meeting the requirements could be easily constructed from a constituency parse tree of the sentence. The set of nodes of the graph is the same as the tree. The set of edges is formed by adding both bottom-up and top-down links between every parent-child pair designated by the tree. The constituent of each leaf node is just one single token, or word, given by the parse. For each internal node, its constituent is the concatenation of the constituents of its children.

doi:10.6342/NTU201704465

The aforementioned **parse tree-derived constituency graph** is presumably the correct constituency structure of the sentence given by a parser or an annotator. However, as we shall see, such a naïve graph might not be optimal for NER and changes could take place.

4.2 Consistency

A named entity does not necessarily correspond to any constituent of a constituency graph.

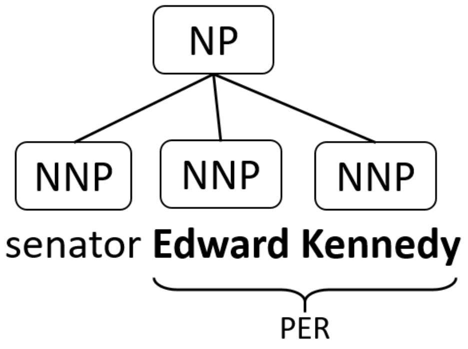
Definition 4.1. If a named entity does not correspond to any node in a given constituency graph, i.e. no single constituent equals the named entity chunk, it is said to be *inconsistent* with the graph.

This happens frequently even for human annotated constituency parse trees. Practically, a given parse tree might not be optimal for NER because of various kinds of inconsistent NEs.

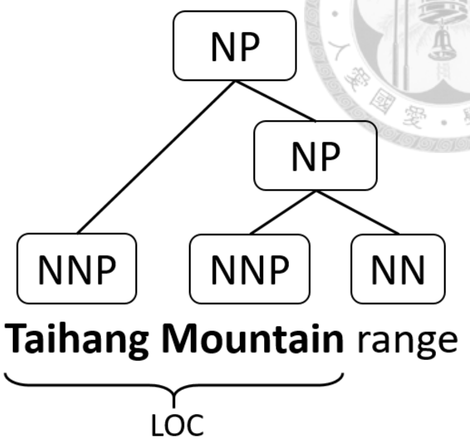
Definition 4.2. An inconsistent named entity is said to be *type-1* inconsistent if it is the concatenation of sibling constituents; *type-2*, otherwise.

So practically, a parse tree-derived constituency graph could be optimized for NER by resolving these inconsistencies. To achieve this without actually knowing NE locations, several algorithms are introduced in the next sections.

doi:10.6342/NTU201704465



(a) Type-1



(b) Type-2

Figure 4.1: Types of inconsistencies between Parses and NER.

4.3 Parse Tree Binarization

4.3.1 Observations of Type-1

Many inconsistent named entities are actually the concatenation of multiple sibling constituents. An example is shown in Figure 4.1a.

For parse tree-derived constituency graphs, type-1 inconsistent named entities only occurs when some nodes have more than two child nodes. Otherwise, suppose a named entity is type-1 inconsistent. The named entity must correspond to the concatenation of two sibling constituents. Because the concatenated chunk must equal to the parent constituent, the named entity is not inconsistent. A conflict.

Further judgement could be made from the above observation: these type-1 inconsistencies might be seen as minor parse errors or just the treebank annotation

doi:10.6342/NTU201704465

style. Although the parser does not group some siblings correctly for NER, it does not group them wrongly. It just does not group them.

4.3.2 Binarizing Parse Trees

Grouping some siblings correctly resolves type-1 inconsistencies. However, NE locations are unknown to the system. Instead, a linguistic-compliant binarization process is applied. Algorithm 1 shows the recursive procedure used by the system. Starting from the root node of a parse tree, the process recursively groups child nodes of which the parent has more than two children. By creating a new child to be the new parent of all original children except only one child to a side, it ensures every node has at most two children.

Essentially, for each node, the head-driven process groups children recursively around the head one, making it the deepest. The heuristic is that a head constituent is usually modified by its siblings in a near to far fashion. Practically, the head child of a node is determined by a rule-based head finder [4]. The finder decides the head for each production, i.e. the parse tags of a node and its children.

4.3.3 An Example

Let a sentence $S_1 = (senator, Edward, Kennedy)$ and a named entity $e_1 = (1, (2, 4), PER)$. Figure 4.2 shows a parse tree-derived constituency graph for S_1. To facilitate discussion, nodes are numbered, and parse tags, head words, and constituents are abbreviated as *pos*, *head*, and *const* respectively. It is clear that

doi:10.6342/NTU201704465

Algorithm 1 Binarization

```
1: function BINARIZE($node$)
2:     $n \leftarrow node.children.length$
3:     if $n > 2$ then
4:         if HEAD-FINDER($node$) $\neq node.children[n]$ then
5:             $newChild \leftarrow$ GROUP($node.children[1..n\text{-}1]$)
6:             $node.children \leftarrow [newChild, node.children[n]]$
7:         else
8:             $newChild \leftarrow$ GROUP($node.children[2..n]$)
9:             $node.children \leftarrow [node.children[1], newChild]$
10:        $newChild.pos \leftarrow node.pos$
11:    for $child$ in $node.children$ do
12:        BINARIZE($child$)
```

e_1 is inconsistent with the graph because no one node corresponds to the chunk $(Edward, Kennedy)$. However, the chunk is actually the constituent concatenation of the siblings *node#3* and *node#4*.

Figure 4.3 shows the application of Algorithm 1 to the parse tree of S_1. With the heuristic that *node#3 (Edward)* modifies the head node *node#4 (Kennedy)* before *node#2 (senator)*. The binarization process successfully adds a new *node#5 (Edward Kennedy)* that corresponds to e_1. In addition, the newly generated child node is given the same parse tag and the head word as its parent.

Effectively, binarizing parse trees eliminates type-1 inconsistencies while leaving

doi:10.6342/NTU201704465

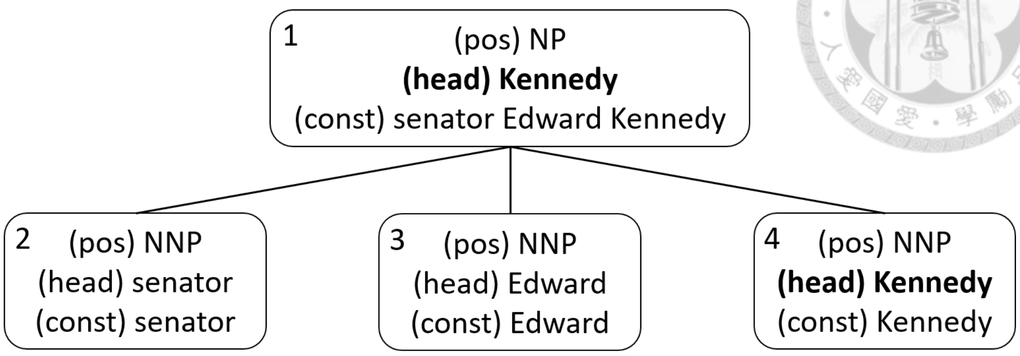


Figure 4.2: A parse tree-derived constituency graph for S_1.

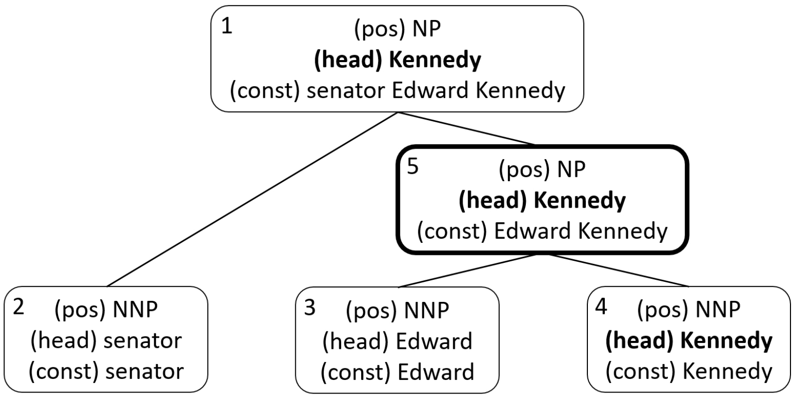


Figure 4.3: Applying Algorithm 1 for S_1.

consistent NEs untouched. In other words, the consistency problem is guaranteed to be mitigated or stay the same, which is extremely unlikely. However, type-1 inconsistent NEs might not be completely resolved, as wrong grouping of siblings only makes some type-1 inconsistent NEs type-2. Figure 4.4 illustrates the situation when *node#3* and *node#4* are not siblings anymore.

doi:10.6342/NTU201704465

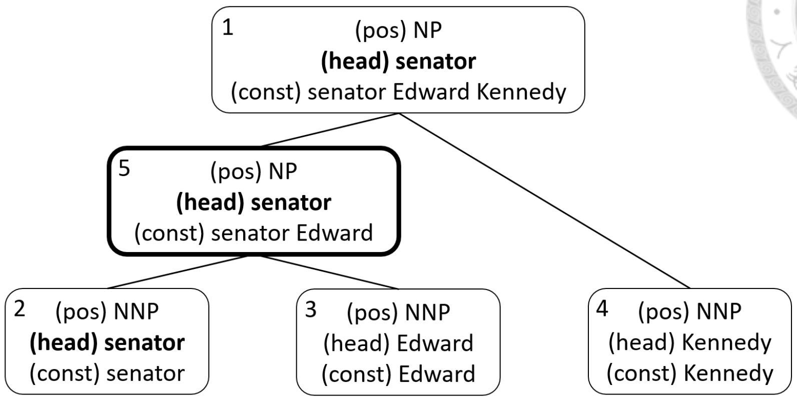


Figure 4.4: Wrong grouping for S_1.

4.4 Pyramid Construction

4.4.1 Observations of Type-2

After binarization, all remaining inconsistent NEs are type-2. This type of inconsistent NEs cross different branches of a parse tree. In other words, a type-2 inconsistent NE is the constituent concatenation of nodes deep down different branches such that they are not siblings. An example is shown is Figure 4.1b.

On one hand, type-2 inconsistences could be seen as major parse errors. In general, every named entity should correspond to a linguistic constituent or at least the combination of some constituents. The parse tree, however, dictates that some needed constituents of an NE should not be combined at all.

On the other hand, type-2 inconsistent NEs could be seen as ungrammatical against the supposedly correct parsing. This happens when a chunk of tokens fits a name well just by chance. Sometimes the writer or the speaker does not intend to

doi:10.6342/NTU201704465

group the tokens of a name but rather group some of them with others first.

For example, in Figure 4.1b, the speaker might just want to use *Taihang* as an adjective for *Mountain range*. However, *Taihang Mountain* fits too well a name not to tag by NER annotators. Note that, in modern Chinese, the name of a mountain is almost always ended by *Mountain* (山). *Taihang Mountain* (太行山) is such a case.

Another example of coincidence is shown in Figure 4.5. The speaker might want to mention the couple by first grouping their first names *Bob and Mary* before their shared last name *Schindler*. But the NER annotators might think that *Mary Schindler* is too good a proper name not to tag.

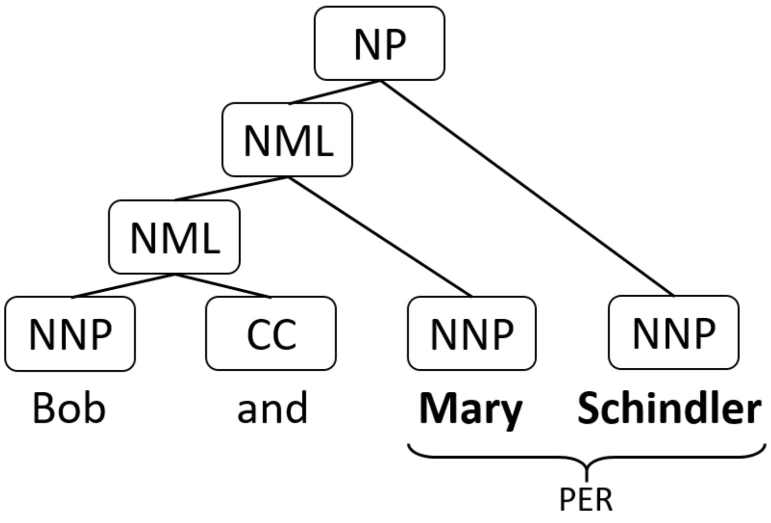


Figure 4.5: Another example of type-2 inconsistency.

Whichever point of view taken, the parser making mistakes or NEs being ungrammatical, this inconsistency could not be resolved solely by the parser. This is

doi:10.6342/NTU201704465

in contrast to the cases of type-1. Without NE information, type-1 inconsistencies are mitigated by using head words determined by parses. For type-2 inconsistencies, trusting the parse is the essence of the problem.

4.4.2 Naïve Pyramids

A so called naïve pyramid has a node for every possible chunking of nodes. Figure 4.6 shows an example. For each node, its constituent is the concatenation of leaf node tokens in the sub-pyramid rooted at the node. While there might be no simple syntactical ways to restructure and in a sense fix the parses without knowing NE locations, this extreme alternative ditches parses and makes sure no inconsistent NEs exist.

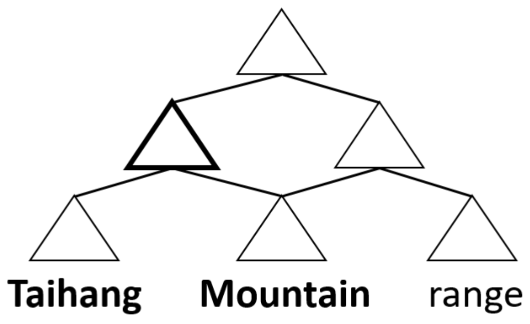


Figure 4.6: A naïve pyramid.

The apparent drawback of a naïve pyramid is that no linguistic structures are present. Instead, it might be better if a parse and a pyramid are combined together into one single constituency graph. Figure 4.7 illustrates the idea.

Still, too much information might just behave like lack of valuable information.

doi:10.6342/NTU201704465

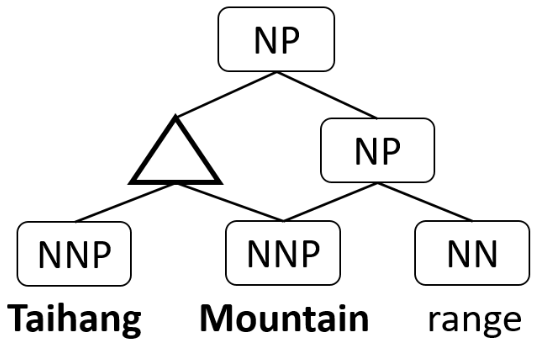


Figure 4.7: A combined graph of a parse tree and a naïve pyramid.

The nodes of a pyramid that do not already exist in parses lack linguistic information such as parse tags and head words. Moreover, the number of nodes explodes when a pyramid is added to a parse tree. Suppose there are n tokens in a sentence. For a parse tree, there will be only n leaf nodes, $n-1$ 2-degree nonterminal nodes, and few 1-degree nonterminal nodes practically. However, for a pyramid, there are $n + (n-1) + (n-2) + \cdots + 1$: more than half of n^2 nodes in total. These overwhelming uninformative new nodes make it much harder for models to learn to propagate structural linguistic information. As a result, phrase structures are diluted too much and training speed becomes unpractically slow.

4.4.3 The Pyramid Addition Method

According to the above reasoning, a novel method is proposed to create a pyramid-added constituency graph. An example is illustrated in Figure 4.8.

First and foremost, the parse tree is preserved by making old links bypass new pyramid nodes. This way, valuable linguistic information is presented to a model in

doi:10.6342/NTU201704465

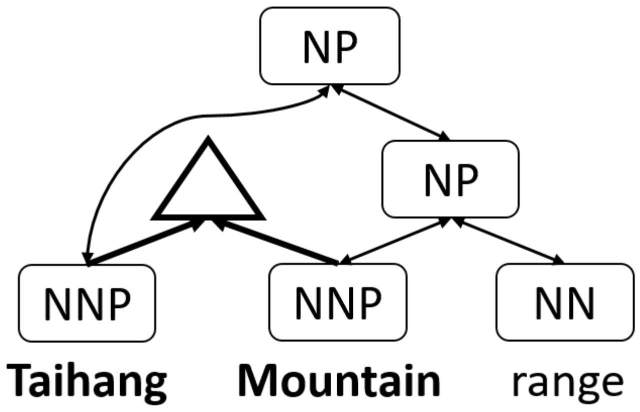


Figure 4.8: The pyrmaid-added constituency graph.

the same structure as the parse-tree derived constituency graph by the untouched parse tree. Newly added pyramid nodes only act as information consumers. Their constituents consume information from original parse tree, but are fed only to other new nodes. As a result, only bottom-up links are added, while previously every line in the illustrations is bidirectional.

Second, the height of the pyramid is limited to a small constant d. Hence the total number of nodes are bounded by $d \times n$, where n is the number of tokens in the sentence. When d is set to 3, all bigrams and trigrams in the sentence correspond to some nodes in the pyramid-added graph.

In summary, the proposed procedure of adding pyramid nodes significantly increases consistency while preserving linguistic structures. This is done by focusing on predicting additional short named entities, which are actually most NEs, with the aid of untouched parse trees.

doi:10.6342/NTU201704465

4.5 Dependency Transformation

An alternative to constituency parsing for constructing a base tree is dependency parsing. However, a dependency parse is not a hierarchy of phrases. Instead, every node in such a parse corresponds to a distinct token and tagged edges give the modification relationship, or dependency, between tokens. Figure 4.9 shows a simple example. Note that it is still a tree, with every arrow pointing from a child to its parent.

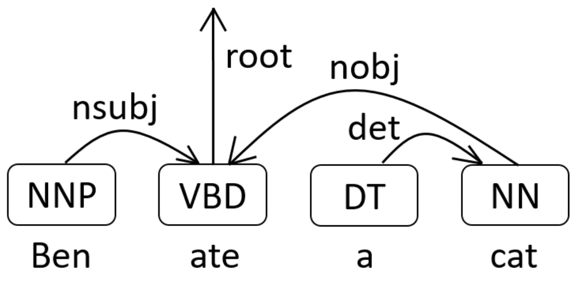


Figure 4.9: A dependency parse tree with bottom-up links.

To use a dependency parse to construct a base tree, a dependency-to-constituency transformation must be applied. Now one strength of a constituency graph is that no strict grammar is required. As long as a consistent hierarchy of nodes is present, underlying models could try to learn from the structures. Algorithm 2 gives the recursive procedure used in the thesis to obtain a hierarchy of token chunks from dependencies.

The process transforms a dependency parse to a constituency parse by recursively making a new root node out of each dependency relation. Originally in a dependency parse, every node corresponds to a distinct token. In the process, this notion is

doi:10.6342/NTU201704465

Algorithm 2 Dependency Transformation

```
1: function TRANSFORM(node)
2:     root ← node
3:     for child in CHILD-QUEUE(node) do
4:         childRoot ← TRANSFORM(child)
5:         if child.token.index < node.token.index then
6:             root ← GROUP([childRoot, root])
7:         else
8:             root ← GROUP([root, childRoot])
9:         root.pos ← RELATION(node, child)
10:    return root
```

generalized to every token corresponds to a subtree headed by it. For each pair of tokens in an unprocessed relation, their current respective subtrees are grouped together by a new root node. The parent token in the original dependency parse then corresponds to the new grown subtree because it is the head token. Figure 4.10 shows the transformed tree of Figure 4.9. A transformed tree is naturally binary, and dependency links determine head child nodes and parse tags.

A detail hidden in CHILD-QUEUE is how the processing order of relations that shared the same head token is decided. In the thesis, the heuristic that a named entity is often centered around a token which is modified in a left-to-right and near-to-far fashion is used. For instance, a noun is often modified in the order of adjectives, a determiner, and a clause.

doi:10.6342/NTU201704465

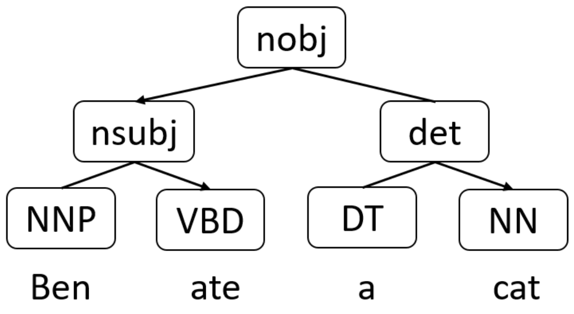


Figure 4.10: The transfromed tree for Figure 4.9. Arrows indicate head nodes.

Effectively, a dependency parser plus the proposed transformation algorithm could take on the role of generating parse tree-derived constituency graphs. This is an alternative to the functional block of constituency parsing in Figure 3.3. The alternative could prove to be useful if a good dependency parser is available.

doi:10.6342/NTU201704465

Chapter 5

Constituent Classification

The second stage of the proposed approach is tasked with classifying constituents by a constituency graph. To utilize relevant constituent structures in classifying each constituent, we propose to use a special recursive network, BRNN-CNN, as the model underlying the stage. By following bottom-up links, the model captures the semantic composition of local information for each constituent. By following top-down links, the global information of the structures containing each constituent is captured. Together, the bidirectional passes propagate relevant information on a constituency graph for the identification of named entity constituents.

5.1 Feature Extraction

For every node in a constituency graph, local features are drawn from itself, its left sibling, and its right sibling. Features in use include words, head words,

doi:10.6342/NTU201704465

and parse tags. However, features are not always available because of the following reasons.

- Absent siblings,
- Absent words for nonterminal nodes, and
- Absent words and head words for added pyramid nodes.

Should these cases happen, dummy feature values are used.

Besides, total number of tokens in a sentence is used as a global feature.

5.1.1 Word-Level Features

For word-level features, a function N_1 is first defined such that it maps each distinct token to a distinct index. Suppose there are n distinct tokens in the corpus. Then their indices are from 1 to n. A special index 0 is mapped for non-existent tokens.

With the index mapping defined, the word-level features are extracted as the following. For each node i with left sibling j and right sibling k, its word-level features

$$x_i = (N_1(word_i), N_1(head_i), N_1(head_j), N_1(head_k))$$

where $word_i$ denotes the word of i, and $head_i$, $head_j$ and $head_k$ denote the head words of i, j, and k respectively.

For example, suppose the word-to-index mapping used is shown in Table 5.1.

doi:10.6342/NTU201704465

For *node#5* in Figure 5.1, its word-level features

$$x_5 = (N_1(word_5), N_1(head_5), N_1(head_2), N_1(head_\phi)$$

$$= (N_1(\phi), N_1(Kennedy), N_1(senator), N_1(\phi))$$

$$= (0, 3, 1, 0)$$

where ϕ represents non-existent nodes and tokens.

x	ϕ	senator	Edward	Kennedy	Bob	and	Mary	Schindler
$N_1(x)$	0	1	2	3	4	5	6	7

Table 5.1: An example of word-to-index mapping. ϕ represents a non-existent token.

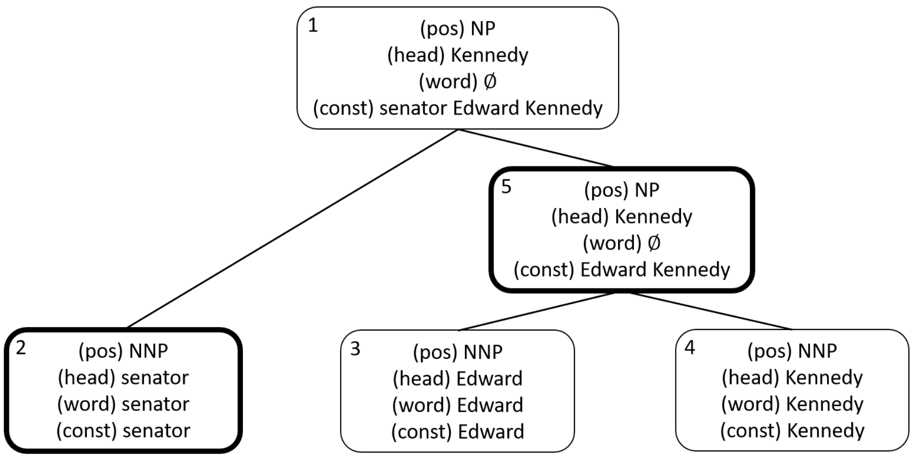


Figure 5.1: Binarized tree for $S_1 = (senator, Edward, Kennedy)$.

doi:10.6342/NTU201704465

5.1.2 Character-Level Features

For character-level features, every word is treated as a sequence of characters, and in the case when a token is non-existent, an empty sequence is used as its character sequence. To facilitate batch computing, words are preprocessed so that they are uniform in length. This is achieved with the aid of a special *padding* character. In addition, special *end* and *start* characters are used to provide boundary information for shift-window models. Algorithm 3 shows the steps of the process. Effectively, for words that are too long, its trailing characters are cut off before prepending *start* and appending *end*. Conversely for those short words, *start* and *end* are added before appending additional *paddings*. The uniform length is set to 20, with which the completeness of most dictionary words are preserved and the noisy tails of long tokens such as web addresses are truncated.

Algorithm 3 Unification

```
1: function UNIFY($word$)
2:     $word \leftarrow [start] + word[1..18] + [end]$
3:     $word \leftarrow word + [padding] \times (20 - word.length)$
4:     return $word$
```

With preprocessed words, a function N_2 is defined such that it maps each distinct character to a distinct index. Suppose there are n distinct characters in the corpus. Then their indices are from 3 to n+2. The indices 0, 1, 2 are reserved for the characters *padding*, *end*, and *start* respectively. For simplicity, the notation is abused so that N_2 could also represent a procedure that takes a character sequence c and returns a sequence of mapped indices of $UNIFY(c)$.

doi:10.6342/NTU201704465

With the procedure defined, the character-level features are extracted as the following. For each node i with left sibling j and right sibling k, its character-level features

$$c_i = (N_2(word_i), N_2(head_i), N_2(head_j), N_2(head_k))$$

where $word_i$ denotes the word of i, and $head_i$, $head_j$ and $head_k$ denote the head words of i, j, and k respectively.

For example, suppose characters $a, \ldots, z, A, \ldots, Z$ are mapped to $3, \ldots, 28, 29, \ldots, 54$ and the uniform length is set to 5 for the purpose of demonstration. Then *senator*, *Kennedy*, and non-existent tokens are unified to *(start,s,e,n,end)*, *(start,K,e,n,end)*, and *(start,end,padding,padding,padding)* respectively. For *node#5* in Figure 5.1, its character-level features

$$\begin{aligned} c_5 &= (N_2(word_5), N_2(head_5), N_2(head_2), N_2(head_\phi) \\ &= (N_2(\phi), N_2(Kennedy), N_2(senator), N_2(\phi)) \\ &= ((2,1,0,0,0), (2,39,7,16,1), (2,21,7,16,1), (2,1,0,0,0)) \end{aligned}$$

where ϕ represents non-existent nodes and tokens.

5.1.3 Parse Tag Features

For parse tag features, a function N_3 is defined such that it maps each distinct parse tag to a distinct index. Suppose there are n distinct parse tags in the grammar. Then their indices are from 1 to n. Non-existent parse tags are mapped to index 0.

For each node i with left sibling j and right sibling k, its parse tag features

$$p_i = (N_3(pos_i), N_3(pos_j), N_3(pos_k))$$

doi:10.6342/NTU201704465

where pos_i, pos_j, and pos_k denote the parse tag of i, j, and k respectively.

For example, suppose the pos-to-index mapping used is shown in Table 5.2. For *node#5* in Figure 5.1, its parse tag features

$$p_5 = (N_3(pos_5), N_3(pos_2), N_3(pos_\phi)$$

$$= (N_3(NP), N_3(NNP), N_3(\phi))$$

$$= (5, 3, 0)$$

where ϕ represents non-existent nodes and parse tags.

p	ϕ	NN	NNS	NNP	NNPS	NP	PYRAMID
$N_3(p)$	0	1	2	3	4	5	6

Table 5.2: An example of pos-to-index mapping. ϕ represents a non-existent parse tag. Added pyramid nodes all have the same sepcial tag.

5.1.4 Lexicon Hit Features

In addition to the aforementioned features derived from parse trees. Additional lexicon hit features are introduced by external lexicon resources. For each node, there is a feature per lexicon. If the constituent of the node can be found in a lexicon, then the lexicon feature value is set to 1 (hit); 0 (not hit), otherwise. All phrases are lower-cased before deciding equality.

For example, suppose there are three lexicons shown in Table 5.3. Then for

doi:10.6342/NTU201704465

node#5 in Figure 5.1, its lexicon hit features $lex_5 = (1, 0, 0)$ because its constituent, *Edward Kennedy*, is found in the first lexicon.

PER	ORG	LOC
donald rumsfeld	european broadcasting union	angelus oaks
edward kennedy	oxford health	california city
finkel	the new york times	london mills
wesley	us airways group inc	waite park

Table 5.3: Three example lexicons of PER, ORG, and LOC.

5.2 BRNN-CNN

A special recursive neural network, BRNN-CNN, is proposed to classify each constituent from a constituency graph. To classify a node, BRNN-CNN does not only consider its features. Instead, BRNN-CNN considers the hidden states of the node, which are recursively computed from the features of its relevant linguistic structures.

To recursively compute hidden states, constituents must be structured as a directed acyclic graph. Then, BRNN-CNN could repeatedly apply its hidden layers from the sources to the sinks of the DAG. For each sentence, two DAGs are formed from its hierarchical constituency graph. One is formed by taking all the nodes and bottom-up links, and the other uses top-down links instead.

doi:10.6342/NTU201704465

Because local information is propagated bottom-up according to constituency structures, bottom-up hidden states capture the semantic composition of each constituent. On the other hand, the features of an ancestry and their siblings are propagated down to each descendent, hence top-down hidden states capture global information for each node. These hidden states of each node together contain the structure information of a sentence relevant to the classification of the corresponding constituent.

5.2.1 Input Layer

The extracted features, described in the previous section, are first processed into real-valued vectors by the input layer of BRNN-CNN. For each node, its feature vector is the concatenation of vectors representing word-level, character-level, parse tag, and lexicon hit features.

Word-Level Vector

Suppose there are n distinct tokens in the corpus and the desired word embedding dimension is d_x. Then BRNN-CNN stores a word embedding look-up table W_x which is a n-by-d_x real-valued matrix. Effectively, every row of W_x represents a word embedding.

Recall that each word-level feature is simply a word index. BRNN-CNN transforms each word index into a n-dimensional one-hot vector except for 0. The special index 0 is transformed to a zero vector. Then the vector is multiplied by W_x to retrieve the word embedding. Finally, for a node i with word-level features x_i, a

doi:10.6342/NTU201704465

vector X_i is formed by concatenating the embeddings of the 4 words in x_i.

For example, suppose $d_x = 2$ and the word embedding look-up table

$$W_x = \begin{pmatrix} 11 & 11 \\ 22 & 22 \\ 33 & 33 \end{pmatrix}.$$

Then the word embedding of word index 3 is computed by

$$\begin{pmatrix} 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 11 & 11 \\ 22 & 22 \\ 33 & 33 \end{pmatrix} = \begin{pmatrix} 33 & 33 \end{pmatrix}.$$

And for word-level features $x_i = (0, 3, 1, 0)$, $X_i = (0, 0, 33, 33, 11, 11, 0, 0)$.

Character-Level Vector

It is more complex to compute a character-level vector for a node than other vectors in the input layer. In a nutshell, BRNN-CNN forms a matrix from the character sequence of each token and put it through a series of convolution, max-pooling, and highway layers. These computations actually consist the CNN part of the model, whereas the so-called input layer is actually the input layer of BRNN.

The first step is to form a character-level feature matrix for each token. Recall that for each token, its character-level features is a sequence of character indices. BRNN-CNN forms a one-hot vector for each character index except for the index 0, which is transformed into a zero vector. These character vectors are then put together into a m-by-n matrix, where m is the uniform word length and n is the number of distinct characters in the corpus plus *end* and *start*.

doi:10.6342/NTU201704465

Then sub-word patterns are captured by putting the character-level feature matrix through multiple convolution kernels in parallel. Kernels might have different heights as their window sizes, but their widths must be n. For example, if a kernel has height h, the convolutional layer will compute an (m-h+1)-by-1 feature map. The feature map will then be max-pooled into a scalar, representing the signal strength of a sub-word pattern of length h. Finally, results of all the kernels are put into a vector, called u, of which length, called d_c, is the number of kernels.

However, as suggested by Kim et al. [15], the CNN-computed character-level feature vector u of each token is put through an additional highway layer. The final character-level vector of a token, called v, is computed as the following.

$$t = \sigma(W_t u + b_t)$$

$$v\prime = ReLU(W_v u + b_v)$$

$$v = (1 - t) \odot u + t \odot v\prime$$

$\sigma(x)$ represents the sigmoid function $\frac{1}{1+e^{-x}}$, and $ReLU(x) = max(0, x)$. W_t and W_v are d_c-by-d_c square weight matrices, and b_t and b_v are d_c-dimensional bias vectors. Essentially, the final vector v is a weighted sum of its input u and the non-linear transformation $v\prime$ of u. By initializing b_t to a negative value, the layer initially sends its input direct to output like a highway.

In summary, for each node i, BRNN-CNN computes a character-level vector for each of the 4 character index sequences in c_i. These 4 vectors are then concatenated to form the character-level vector of the node, called C_i, for the input layer.

doi:10.6342/NTU201704465

Parse Tag Vector

A parse tag vector is computed for each node i. Suppose there are d_p distinct parse tags in the corpus. For each parse tag index in p_i, a d_p-dimensional one-hot vector is formed. The exception is that index 0 is transformed into a zero vector. These vectors are concatenated into a long vector, called P_i, for the input layer.

Global Word Feature Vector

Aside from a constituent itself and its ancestors, the whole sentence provides useful additional information for NER. BRNN-CNN averages the word-level vectors of all tokens in a sentence as M_x. Similarly, the character-level vectors of all the tokens in a sentence are averaged to get another mean embedding M_c. These two vectors are used for every node in the constituency parse of the sentence as global knowledge.

Input Layer Vector

Finally, for each node i, BRNN-CNN computes its input layer by Equation 5.1.

$$I_i = X_i \| C_i \| P_i \| lex_i \| M_x \| M_c. \tag{5.1}$$

The dimension of I_i is given by

$$d_I = d_x \times 4 + d_c \times 4 + d_p \times 3 + d_{lex} + d_x + d_c$$

where d_{lex} is the number of lexicons.

doi:10.6342/NTU201704465

5.2.2 Hidden Layers

Having computed the input layer for every node on a constituency graph, BRNN-CNN recursively computes two hidden states for every node.

For each node i with the set of child nodes N and the parent p, the bottom-up hidden vector $H_{bot,i}$ and top-down hidden vector $H_{top,i}$ are recursively computed by Equation 5.2 and Equation 5.3 respectively.

$$H_{bot,i} = ReLU((I_i \| \sum_{j \in N} H_{bot,j})W_{bot} + b_{bot}) \tag{5.2}$$

$$H_{top,i} = ReLU((I_i \| H_{top,p})W_{top} + b_{top}) \tag{5.3}$$

The function $ReLU(x)$ represents $max(0, x)$. Suppose d_H is the desired hidden feature dimension. Then W_{bot} and W_{top} are (d_I+d_H)-by-d_H weight matrices. b_{bot} and b_{top} are d_H-dimensional bias vectors. If the set of child nodes are empty, a d_H-dimensional zero vector is used instead of $\sum_{j \in N} H_{bot,j}$. Similarly if i has no parent, a zero vector is used instead of $H_{top,p}$.

The computations of the bottom-up and top-down hidden states of *node#5* in Figure 5.1 are illustrated in Figure 5.2 and 5.3. The full bottom-up and the top-down passes on the constituency graph in Figure 5.1 are illustrated in Figure 5.4 and Figure 5.5.

doi:10.6342/NTU201704465

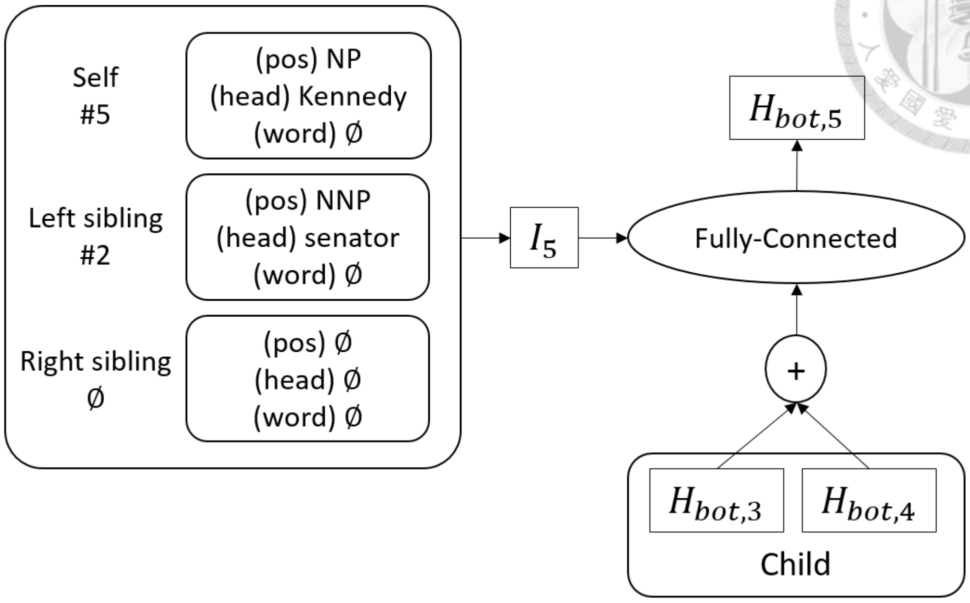


Figure 5.2: The bottom-up hidden layer applied to *node#5* in Figure 5.1.

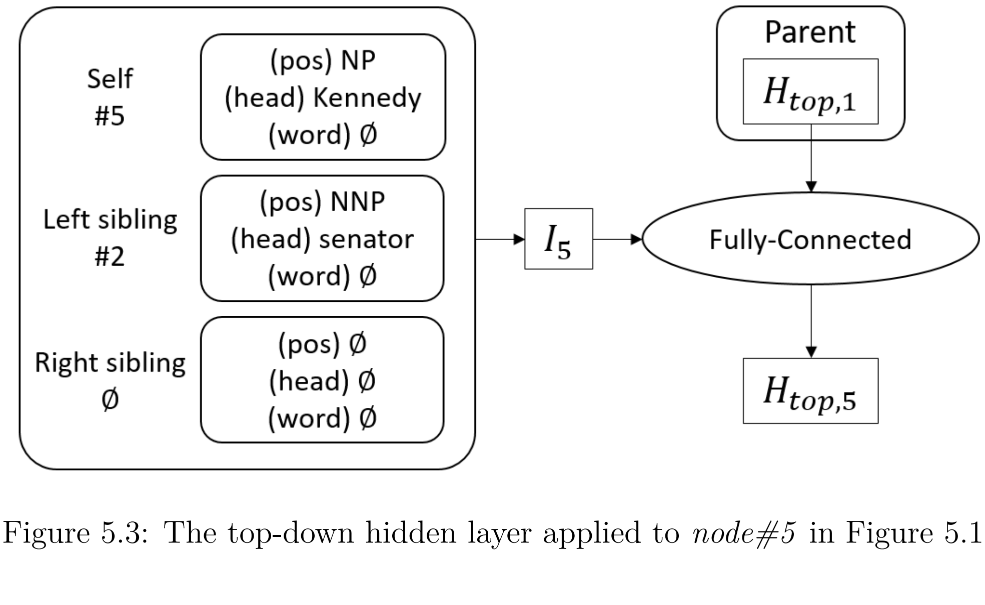


Figure 5.3: The top-down hidden layer applied to *node#5* in Figure 5.1.

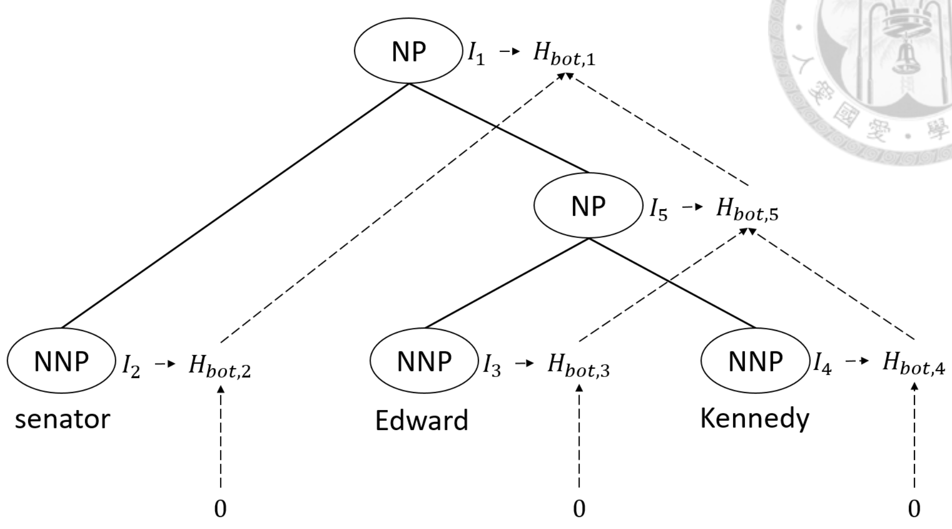

NP
$I_1 \rightarrow H_{bot,1}$
NP
$I_5 \rightarrow H_{bot,5}$
NNP
$I_2 \rightarrow H_{bot,2}$
senator
NNP
$I_3 \rightarrow H_{bot,3}$
Edward
NNP
$I_4 \rightarrow H_{bot,4}$
Kennedy
0
0
0

Figure 5.4: The bottom-up hidden layer applied to the graph in Figure 5.1.

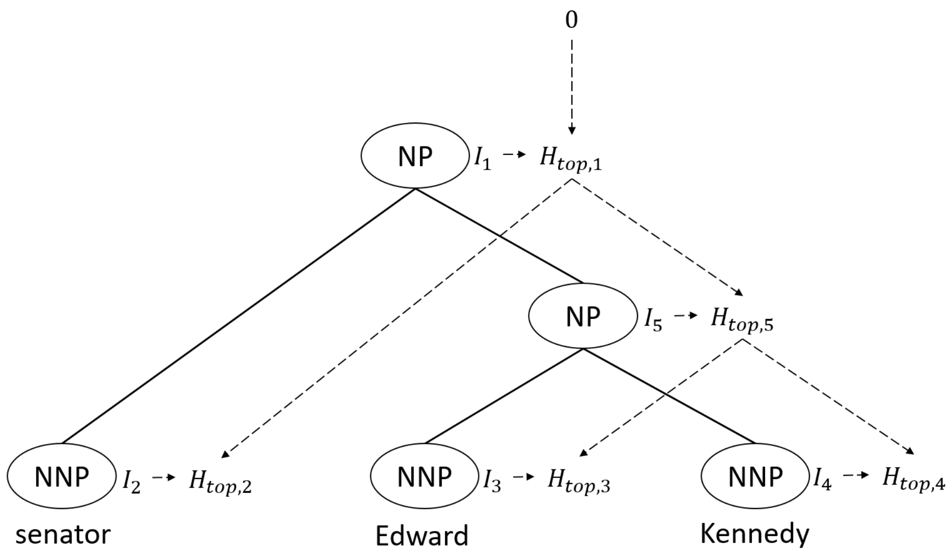

0
NP
$I_1 \rightarrow H_{top,1}$
NP
$I_5 \rightarrow H_{top,5}$
NNP
$I_2 \rightarrow H_{top,2}$
senator
NNP
$I_3 \rightarrow H_{top,3}$
Edward
NNP
$I_4 \rightarrow H_{top,4}$
Kennedy

Figure 5.5: The top-down hidden layer applied to the graph in Figure 5.1.

doi:10.6342/NTU201704465

Deep Hidden Layers

There can be more than one recursive hidden layer in BRNN-CNN. One-layered recursive neural networks is already deep in the sense of recursion: the hidden layer is stacked as many times as the height of the input hierarchy. Having multiple hidden layers, however, makes more powerful transformations between two neighboring nodes possible.

Suppose there are 3 hidden layers with desired dimension d_{H1}, d_{H2}, and d_{H3}. For each node i with parent node p, the top-down hidden state features can be computed by the following.

$$H_{top,i,1} = ReLU((I_i \| H_{top,p,1})W_{top,1} + b_{top,1})$$

$$H_{top,i,2} = ReLU((H_{top,i,1} \| H_{top,p,2})W_{top,2} + b_{top,2})$$

$$H_{top,i,3} = ReLU((H_{top,i,2} \| H_{top,p,3})W_{top,3} + b_{top,3})$$

$ReLU(x) = max(0, x)$. $W_{top,1}$, $W_{top,2}$, and $W_{top,3}$ are (d_I+d_{H1})-by-d_{H1}, (d_{H1}+d_{H2})-by-d_{H2}, and (d_{H2}+d_{H3})-by-d_{H3} weight matrices respectively. $b_{top,1}$, $b_{top,2}$ and $b_{top,3}$ are d_{H1}, d_{H2}, and d_{H3}-dimensional bias vectors respectively. Figure 5.6 illustrates the three-layered computation. The bottom-up direction is generalized similarly.

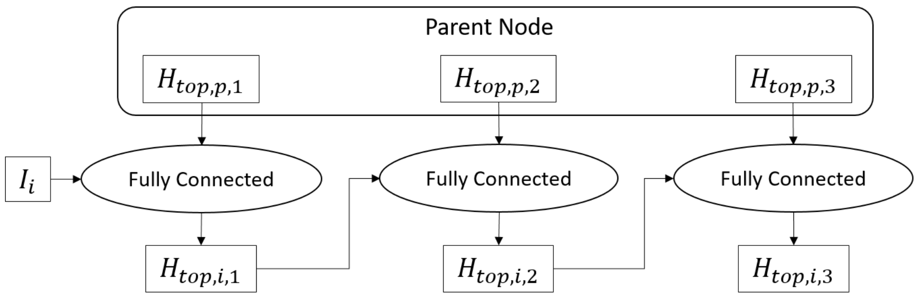


Figure 5.6: The top-down *deep* hidden layer applied to node i with parent p.

doi:10.6342/NTU201704465

5.2.3 Output Layer

The output layer of BRNN-CNN identifies named entity constituents. For each node, the input bottom-up and top-down hidden state vectors contain relevant local and global information of the corresponding constituent. The output is the predicted probability distribution of named entity classes.

Given the set of named entity categories C with size n, a function N_4 is first defined to map each distinct NE category to a distinct integer between $1 \dots n$. The number $n + 1$ is reserved for the special category *NON_NE*. The inverse function mapping integers to categories is denoted by N_4^{-1}. Therefore the dimension of predicted distributions $d_O = n + 1$.

For any node x, let $H_x = H_{bot,x} + H_{top,x}$. And let σ denote the softmax function where, for any vector x, $\sigma(x)_t = \frac{e^{x_t}}{\sum\limits_u e^{x_u}}$. Then for each node i with left sibling j and right sibling k, its class probability distribution is given by Equation 5.4.

$$O_i = \sigma((H_i \| H_j \| H_k) W_{out} + b_{out}) \tag{5.4}$$

W_{out} is a $(d_H \times 3)$-by-d_O weight matrix. b_{out} is a d_O-dimensional bias vector. If a sibling does not exist, zero vectors are used as its hidden states. Should deep hidden layers be deployed, the last hidden layer is used. Figure 5.7 illustrates the computation of the output layer.

doi:10.6342/NTU201704465

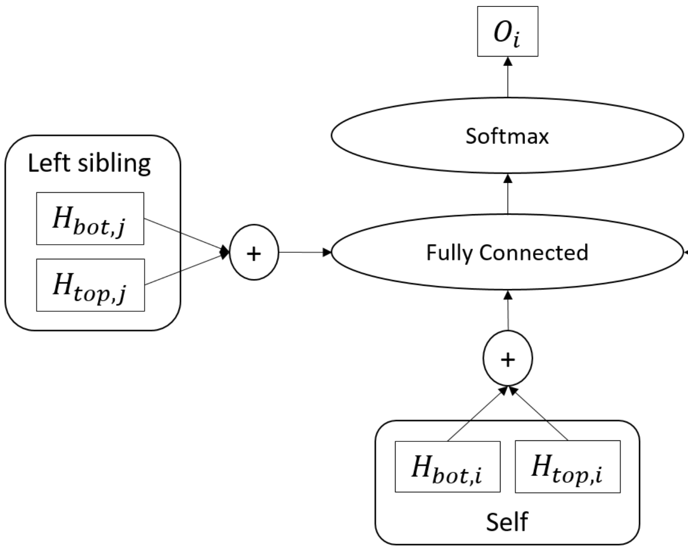


Figure 5.7: The output layer applied to node i with left sibling j and right sibling k. The fully-connected layer does not contain non-linear transformations like ReLU.

5.3 Prediction Collection

After BRNN-CNN produces a probability distribution for each constituent, the set of predicted named entities are now collected from the constituency graph.

For each constituent, the system classifies it as the category with highest predicted probability. Formally, for each node i, the system predicts its label by Equation 5.5.

$$L_i = N_4^{-1}(\underset{j}{\operatorname{argmax}}\, O_{ij}) \tag{5.5}$$

O_{ij} represents the j-th element of O_i.

Intuitively, the predictions of a sentence are the label L_i for every node i unless $L_i = NON\_NE$. Given the set of sentences S and the set of named entity categories

doi:10.6342/NTU201704465

C, the set of predicted named entities

$$E_a = \{(i, (j, k), L_n) \mid S_i \in S, n \in GN_{S_i}, (S_{ij}, \ldots, S_{i(k-1)}) = const_n, L_n \in C\}$$

where GN_{S_i} denotes the set of nodes in the constituency graph of the sentence S_i, and $const_n$ denotes the corresponding constituent of the node n.

In many applications, overlapping named entities, including nested named entities, are not practically useful and should not be predicted by NER systems. These could happen to the above NE collection method. For instance, two NEs are nested if their corresponding nodes are an ancestor and its descendant.

To form non-overlapping predictions, a special NE collection scheme is applied by the system. The scheme traverses through every node of a constituency graph in two passes. The first pass performs a depth-first walk from the root node of the original constituency parse tree in the graph. The system stops recurring down a branch as soon as it encounters a node i in the branch of which $L_i \neq NON\_NE$. Then a second pass walk through the additional pyramid nodes in a breadth-first fashion, collecting an additional NE only if it does not overlap with previously collected NEs.

Briefly speaking, the system forms non-overlapping predictions from the output of BRNN-CNN by preferring tree nodes over additional pyramid nodes and larger NEs over smaller ones. The heuristics behind this scheme is to appreciate parsing and to avoid nested NEs, like the first name and the last name of a full name.

doi:10.6342/NTU201704465

Chapter 6

Evaluation

The constituency-oriented approach is evaluated on CoNLL 2003 NER and OntoNotes 5.0 NER. The detailed setup of the experiments is given in the first section. Major results against state-of-the-art systems and related work are shown in the second section. Finally, we analyze different aspects of the approach with ablation studies and discuss their abstractive meaning with case studies.

6.1 Experimental Setup

6.1.1 Parameter Initialization

Parameters of BRNN-CNN are contained in the CNN, the highway layer, the BRNN hidden layers, and the output layer. In addition, the word embedding look-up table is also trainable. Table 6.1 summarizes the weights and biases of which values need to be decided.

doi:10.6342/NTU201704465

Except for W_x, the parameters of all other layers are initialized with Xavier initializer [11]. The initializer tries to ensure the scale of output values for deep networks. This is desirable because BRNN-CNN might go down an indefinitely deep recursion.

On the other hand, the initialization of the word embedding look-up table has two cases. First, a pre-trained table is attempted to be used. For example, unsupervised word embeddings trained by GloVe [23] from a 840 billion-token web corpus are available. If a pre-trained GloVe vector could be found for a word, the corresponding row of W_x is initialized by that vector. Otherwise, the Gaussian distribution with zero mean and 0.1 standard deviation is used for sampling.

Layer	**Weights and Biases**
Word Embedding Look-Up Table	W_x
CNN	Kernels
Highway Layer	W_t, W_v, b_t, b_v
Hidden Layers	W_{bot}, W_{top}, b_{bot}, b_{top}
Output Layer	W_{out}, b_{out}

Table 6.1: Parameters of BRNN-CNN.

6.1.2 Parameter Optimization

Once initialized, the model parameters can be optimized with a training corpus. In a training corpus, all ground truth named entities are known to BRNN-CNN.

doi:10.6342/NTU201704465

For each node n in the constituency graph of a sentence, its ground truth NE label is denoted by $L_{n,g}$. If n corresponds to a named entity $e = (i,(j,k),c)$, $L_{n,g} = c$. Otherwise, $L_{n,g} = NON\_NE$. Equation 6.1 gives the loss function of a node.

$$loss_n = -\log O_{nN_4(L_{n,g})} \tag{6.1}$$

$O_{nN_4(L_{n,g})}$ denotes the $N_4(L_{n,g})$-th element of O_n. The objective is set to minimize the average loss of all nodes for a corpus. To achieve this, Adam optimizer [16] is used for parameter updates.

To avoid overfitting for gradient descent optimization algorithms such as Adam, it is desirable to stop iterating parameter updates with the aid of a validation corpus. Let F denotes an evaluation function, E_a denotes the prediction of BRNN-CNN for the validation corpus, and E_g denotes the set of ground truth NEs of the corpus. After each round, or epoch, of parameter update with the training corpus, the validation score $F(E_a; E_g)$ is checked. Since initialization, the best score is kept. If a record has not been broken for 20 epochs, the training stops and the parameter values that achieved the best score are restored.

Dropout

Zeroing some of the output of some network layers is oftentimes beneficial to training. BRNN-CNN is no exception and dropout layers are added for the input layer and the hidden layers. For each node i, elements of I_i (excluding lex_i), $H_{bot,i}$, and $H_{top,i}$ are randomly zeroed in training time. The dropout probability must be carefully chosen so the dropout layers have due effects.

doi:10.6342/NTU201704465

6.1.3 CoNLL-2003 Dataset

In 2003, CoNLL held a language-independent NER shared task. This classic dataset annotates 4 types of NEs: PER (person), ORG (organization), LOC (location), and MISC (miscellaneous). Its English corpus, the Reuters Corpus Volume 1 (RCV1), contains about 300,000 tokens and near 35,000 named entities. A standard train-validate-test split is defined by the shared task.

However, CoNLL-2003 dataset contains no parse annotations. Readily available parsers are used for BRNN-CNN instead. One of the parses used is the Stanford RNN parser [27], which is also a recursive network model. The other is the SyntaxNet dependency parser [1] recently released by Google. The transformation algorithm in Chapter 4 is applied to the dependency parses generated by SyntaxNet.

Table 6.2 shows the dataset statistics with transformed SyntaxNet parses before adding additional pyramid nodes. Even with SyntaxNet, the 93% consistency rate, which equals the maximal possible recall, is still not ideal.

Split	**Sentences**	**Tokens**	**Nodes**	**NEs**	**Consistent NEs**
Train	14,041	203,621	393,201	23,326	21,816 (93.53%)
Validate	3,250	51,362	99,474	5,902	5,440 (92.17%)
Test	3,453	46,435	89,417	5,613	5,197 (92.59%)

Table 6.2: Dataset statistics for CoNLL-2003.

doi:10.6342/NTU201704465

6.1.4 OntoNotes 5.0 Dataset

OntoNotes is a project which creates substantially larger corpora from multiple sources with multilevel annotations. The dataset was used for the CoNLL-2012 shared task of coreference resolution and are increasingly popular as a benchmark for NE-related tasks. The 5.0 release boasts various data sources such as newswire, broadcast conversation, and web text. For NER, 18 categories are annotated, including 11 types of names and 7 types of numerical values. Its English corpus contains near 1,400,000 tokens and more than 100,000 NEs. A standard train-validate-test split is widely used and described in 2013.

OntoNotes comes with both gold-standard parses and automatically generated parses. However, even the gold-standard parses are sometimes inconsistent with NER annotations. The binarization procedure introduced in Chapter 4 is applied to both type of parses to further boost their consistency with NER.

Table 6.3 shows the dataset statistics with binarized auto parses without adding additional pyramid nodes. The 97% consistence rate still hinders constituent-based approaches a bit, but is already desirable.

Split	**Sentences**	**Tokens**	**Nodes**	**NEs**	**Consistent NEs**
Train	59,920	1,088,503	2,388,362	81,828	79,636 (97.32%)
Validate	8,527	147,724	324,841	11,066	10,735 (97.01%)
Test	8,262	152,728	335,101	11,257	10,936 (97.15%)

Table 6.3: Dataset statistics for OntoNotes 5.0.

doi:10.6342/NTU201704465

6.1.5 Hyper-Parameters

Before training BRNN-CNN, many hyper-parameters remain to be determined. This includes the dimensions of layers, initialization options, and parameters of optimization algorithms. Good values of hyper-parameters often depend on each corpus, and they are determined by the validation split with multiple training trials. Table 6.4 summarizes the hyper-parameters and tried values.

Hyperparameter	**Trial Range**	**Final Setting**	
		CoNLL-2003	OntoNotes
Input Layer			
Pre-Trained Word Vectors	50d-Collobert, 300d-GloVe	300d-GloVe	
Character Vector Dimension	number of characters (one-hot), 25	number of characters	
Maximal Kernel Height	3, 5	3	
Kernels for Each Height	$h \times 20$, $h \times 40$	$h \times 40$	
Lexicons (Non-Exclusive)	SENNA, DBpedia	SENNA	
Hidden Layers			
Hidden Layer Type	fully-connected, Tree-LSTM	fully-connected	
Number of Hidden Layers	1, 2, 3	1	3
Hidden Vector Dimension	300, 350, 400, 450, 500, 600, 1200	450	350
Optimization			
Learning Rate of Adam	1e-5, 1e-4, 1e-3	1e-5	
Epsilon of Adam	1e-8, 1e-4, 1e-2, 1e-1, 1e-0	1e-2	
Keep Rate of Dropout	0.65, 0.70, 0.75, 0.80, 0.90, 1	0.65	

Table 6.4: Trial range and final settings of hyper-parameters.

For the word-level vectors, the word vector dimension is decided alongside the

doi:10.6342/NTU201704465

pre-trained vectors used for initialization. One set of the pre-trained word vectors is the 300-dimensional vectors trained by GloVe on an 840 billion-token web corpus. The other is the 50-dimensional vectors of lower-cased tokens released by Collobert in his SENNA system [5].

For character-level vectors, one important thing is how a vector is generated for each character. As stated in Chapter 5, one-hot vector can be used. However, a trainable character embedding look-up table could also be used if the corpus is large enough. As for the kernels, there are 40, 80, and 120 kernels for each height 1, 2, and 3.

For lexicon features, which external resources are used need to be decided. The lexicons released with the SENNA system are tailored for the four categories of the CoNLL-2003 dataset, so they are particularly useful for that dataset. In addition, the PER, ORG, and LOC named entity categories of CoNLL-2003 coincide with some of the categories of OntoNotes, so the three are also good for OntoNotes. On the other hand, lexicons can be extracted from DBpedia [18], the ontology of Wikipedia. For example, a lexicon of person names could be constructed by including every instance falling into a type in the subtree rooted by the *person* type.

For hidden layers, simple fully-connected layers work better than tree-LSTM cells. Its best dimensions depend on the number of layers. In Table 6.4, 350 is selected as the hidden dimension for deep hidden layers, but 450 is selected if there is only one hidden layer per direction.

In the thesis, the final values of these hyper-parameters are mostly searched greedily by the validation data of OntoNotes. A better configuration for each corpus

doi:10.6342/NTU201704465

could very likely be found by a comprehensive grid search.

6.2 Major Results

In this section, the major results on the CoNLL-2003 dataset and the OntoNotes dataset are reported against previous state-of-the-arts and related approaches.

As previous work on these datasets, the selected evaluation function for all the results is the F1 score. While precision measures the proportion of correct predictions over all predicted NEs, recall measures the proportion of correct predictions over all ground truth NEs. F1 scores encourage a balance between the two by taking the harmonic mean.

Table 6.5 and Table 6.6 shows the results and comparisons of the proposed constituency-oriented approach on CoNLL-2003 and OntoNotes respectively. On CoNLL-2003, the proposed approach achieves near state-of-the-art results. On OntoNotes, the proposed approach surpasses the state-of-the-art as well as previous work on joint models of NER and parsing or other NE related tasks.

The significance of the results is computed in the following. The sample mean, standard deviation, and sample count of BRNN with auto parses and Chiu and Nichols' model are (87.10, 0.14, 3) and (86.41, 0.22, 10) respectively. By the conservative one-tailed Welch's T-test, the former significantly surpasses the latter with 99% confidence level (0.000489 p-value).

doi:10.6342/NTU201704465

			Validation			**Test**		
Model	Parser	Pyramid	Precision	Recall	F1	Precision	Recall	F1
BRNN	StanfordRNN	No	93.5	86.8	90.04	89.6	82.2	85.73
BRNN	SyntaxNet	No	92.6	86.4	89.40	89.1	82.9	85.89
BRNN	StanfordRNN	Yes	93.0	91.6	92.34	88.9	86.9	87.91
BRNN	SyntaxNet	Yes	93.1	91.6	92.33	90.2	87.7	**88.91**
Chiu and Nichols [3]			-	-	-	91.39	91.85	**91.62**

Table 6.5: Experiment results on CoNLL-2003.

			Validation			**Test**		
Model	Parser	Pyramid	Precision	Recall	F1	Precision	Recall	F1
BRNN	auto	No	86.0	84.7	85.34	88.0	86.2	87.10
BRNN-CNN	auto	No	85.5	84.7	85.08	88.0	86.5	**87.21**
BRNN	gold	No	87.5	86.7	87.11	89.5	88.3	88.91
BRNN-CNN	gold	No	86.6	87.0	86.77	88.9	88.9	88.92
Finkel and Manning [10] (gold*)			-	-	-	84.04	80.86	82.42
Durrett and Klein [9]			-	-	-	85.22	82.89	84.04
Chiu and Nichols [3]			-	-	-	-	-	**86.41**

Table 6.6: Experiment results on OntoNotes. *Finkel and Manning used gold parses in training time.

doi:10.6342/NTU201704465

Results on Different Types of Data

The OntoNotes 5.0 corpus contains NER annotations for six types of data: broadcast conversation (BC), broadcast news (BN), magazine (MZ), newswire (NW), telephone conversation (TC), and web (WB). These data sources greatly differ from each other and could have their own application and research domain.

Table 6.7 shows the performance of the constituency-oriented BRNN against previous state-of-the-art models. BRNN outperforms previous work from the most well-structured news text to the noisiest telephone conversations, but is less successful in magazine text. We hope the experiments shed light for further studies on these different domains.

	BC	**BN**	**MZ**	**NW**	**TC**	**WB**
Tokens	32,488	23,209	17,875	49,235	10,976	18,945
NEs	1,697	2,184	1,163	4,696	380	1,137
Consistency Rate*	96.29%	98.12%	97.42%	97.19%	96.84%	96.22%
Finkel and Manning [10]	78.66	87.29	82.45	85.50	67.27	72.56
Durrett and Klein [9]	78.88	87.39	82.46	87.60	72.68	76.17
Chiu and Nichols [3]	85.23	89.93	**84.45**	88.39	72.39	78.38
BRNN	**85.73**	**90.63**	83.92	**89.15**	**73.08**	**80.05**

Table 6.7: Experiment results on different data sources of OntoNotes. *Percentage of NEs that correspond to some constituents in binarized auto parses.

doi:10.6342/NTU201704465

6.3 Analysis

6.3.1 Constituency-Oriented Approach

Filtering out non-constituent text chunks is both the strength and the limitation for every constituent-based model. Focusing on constituents allows better precision at the cost of limiting recall to the proportion of consistent named entities.

Looking into this aspect, a sequential labeling bidirectional recurrent network is experimented on OntoNotes. This model takes no phrase structures as input, but all non-constituent named entity predictions can be removed in post-processing. Table 6.8 shows the performance of the sequential labeling model with and without post-processing. As is expected, precision increases and recall decreases with post-processing. Interestingly, we see a marginal overall F1 score improvement when the sequential labeling model filters out non-constituent predictions in post-processing.

The experiment indicates that better results could be achieved if constituents were more consistent with NER or if the full information of phrase structures was utilized. This is when our constituency-oriented approach steps in, fulfilling both requirements with graph generation algorithms and BRNN.

Figure 6.1 shows the parse tree of a sample sentence with the named entity chunk *White House*. The sequential labeling recurrent model predicts a false positive chunk *the White*. The post-processing is able to remove the non-constituent prediction, but the correct named entity cannot be recovered. Finally, our constituency-oriented BRNN fully utilizes the phrase structure information and correctly identifies the node ($NP,\ const = White\ House$) as a *FACILITY*.

doi:10.6342/NTU201704465

		Validation			**Test**		
Model	Const-Only	Precision	Recall	F1	Precision	Recall	F1
Recurrent	No	84.6	85.5	85.03	85.7	**86.5**	86.10
Recurrent	Yes	86.0	84.2	85.08	**87.2**	85.1	**86.14**
Recursive (BRNN)	Yes	86.0	84.7	85.34	**88.0**	86.2	**87.10**

Table 6.8: Performance of sequential labeling models on OntoNotes.

```
the first couple moves out of the White House on January 20th .

|--S
    |--S
    |   |--NP
    |   |   |--DT the
    |   |   |--NP
    |   |       |--JJ first
    |   |       |--NN couple
    |   |
    |   |--VP
    |       |--VP
    |       |   |--VBZ moves
    |       |   |--PP
    |       |       |--IN out
    |       |       |--PP
    |       |           |--IN of
    |       |           |--NP
    |       |               |--DT the
    |       |               |--NP
    |       |                   |--NNP White
    |       |                   |--NNP House
    |       |
    |       |--PP
    |           |--IN on
    |           |--NP
    |               |--NNP January
    |               |--NN 20th
    |
    |--. .
```

Figure 6.1: The parse tree of a sentence containing *White House (FACILITY)*.

doi:10.6342/NTU201704465

6.3.2 Constituency Graph Generation

One strength of the approach is the flexibility of constituency graphs. Here we analyze the proposed algorithms that alter parses and create structures that are more consistent with a targeted NER task.

Parse Tree Binarization

The parse tree binarization method introduced in Chapter 4 is shown to be effective. In Table 6.9, both the auto parses and the gold parses of OntoNotes have their consistency rate greatly increased. As a result, the performance also greatly increases (Table 6.10).

	Auto Parse			Gold Parse		
	Train	Validate	Test	Train	Validate	Test
Original	93.26	92.78	92.91	95.09	94.89	95.07
Binarized	97.32	97.01	97.15	98.66	98.71	98.56

Table 6.9: Consistency rates before and after binarization.

			Validation			**Test**		
Model	Parser	Binarize	Precision	Recall	F1	Precision	Recall	F1
BRNN	auto	No	84.9	81.6	83.20	87.3	83.0	85.11
BRNN	auto	Yes	86.0	84.7	85.34	**88.0**	**86.2**	**87.10**

Table 6.10: Performance on OntoNotes before and after binarization.

doi:10.6342/NTU201704465

Pyramid Addition

The pyramid addition method is proved to be useful when parses are lackluster. For example, the validation recall with SyntaxNet increases greatly from 86.4% to 91.6%. This is because that adding additional pyramid nodes to transformed SyntaxNet parses increases the validation set consistency to 99%.

Dependency Transformation

Dependency parses transformed by simple heuristics could be better than classical constituency parsers in assisting NER. In Table 6.5, using transformed SyntaxNet dependency parses as base parse trees outperforms using StanfordRNN constituency parses.

6.3.3 Constituent Classification

For the classification stage, BRNN-CNN is proposed to capture the structure information presented in constituency graphs. Here, we analyze different parts of the proposed recursive model.

Character-Level Features

BRNN is BRNN-CNN without character-level features. In other words, its input layer is deprived of C_i and M_c of Equation 5.1. As shown in Table 6.6, BRNN-CNN performs better than pure BRNN when the corpus is big enough to train the CNN. However, BRNN runs about twice as fast, so we run most analyses with it.

doi:10.6342/NTU201704465

Bidirectional Hidden Layers

The bottom-up hidden states capture the local structures inside each constituent. On the other hand, the top-down hidden states capture the global structures outside each constituent. Table 6.11 shows the performance differences when only one pass is deployed. Compared to the bottom-up-only variant, BRNN sees a 1.4% precision increase with the additional top-down pass.

	Validation			Test		
Model	Precision	Recall	F1	Precision	Recall	F1
Top-Down-Only	78.2	69.4	73.49	79.2	69.3	73.93
Bottom-Up-Only	84.5	84.3	84.39	86.6	86.2	86.41
Bidirectional	86.0	84.7	85.34	**88.0**	**86.2**	**87.10**

Table 6.11: Performance of unidirectional and bidirectional models on OntoNotes.

Figure 6.2 shows one of the cases where a top-down pass helps. The leaf node $(NNP,\ word = Koran)$ is labeled as *WORK_OF_ART*. Unfortunately, Koran is in the *PERSON* lexicon. Seeing only the local features of the leaf node, the bottom-up-only variant wrongly predicts the constituent as a named entity of type *PERSON*. With the top-down information of ancestor nodes like $(NP,\ head = verses)$ and $(VP,\ head = repeating)$, BRNN correctly identifies $(NNP,\ word = Koran)$ as a *WORK_OF_ART*.

doi:10.6342/NTU201704465

```
He confirmed it by repeating verses from the noble Koran
and the two testimonies .

S
|--S
|   |--NP
|   |   |--PRP He
|   |
|   |--VP
|       |--VP
|       |   |--VBD confirmed
|       |   |--NP
|       |       |--PRP it
|       |
|       |--PP
|           |--IN by
|           |--S
|               |--VP
|                   |--VBG repeating
|                   |--NP
|                       |--NP
|                       |   |--NP
|                       |   |   |--NP
|                       |   |   |   |--NNS verses
|                       |   |   |
|                       |   |   |--PP
|                       |   |       |--IN from
|                       |   |       |--NP
|                       |   |           |--DT the
|                       |   |           |--NP
|                       |   |               |--JJ noble
|                       |   |               |--NNP Koran
|                       |   |
|                       |   |--CC and
|                       |
|                       |--NP
|                           |--DT the
|                           |--NP
|                               |--CD two
|                               |--NNS testimonies
|
|--. .
```

Figure 6.2: The parse tree of a sentence containing *Koran (WORK_OF_ART)*.

doi:10.6342/NTU201704465

Chapter 7

Conclusion

We have proposed a constituency-oriented approach for named entity recognition, where we generate a constituency graph before identifying named entity constituents in the graph. In the construction of a more general hierarchy of constituents, number of inconsistent named entities are minimized while linguistic structures are preserved. In the classification of constituents, the power of prior linguistic structure information is leveraged.

7.1 Summary of Contributions

To optimize the constituency structures for NER, we have defined two types of inconsistent named entities that cross constituent boundaries and proposed various methods to minimize their numbers. Then, to actually utilize the structure information for NER, we have proposed the recursive model BRNN-CNN. Specific

doi:10.6342/NTU201704465

contributions include:

- Definition of two types of inconsistent named entities that cross constituent boundaries,
- Elimination of type-1 inconsistencies by a head-driven binarization process that alters a constituency parse,
- Elimination of type-1 inconsistencies by a relation-based transformation algorithm that restructures a dependency parse,
- Mitigation of type-2 inconsistencies by a proposed method that augments binarized trees,
- Utilization of relevant local structures for each constituent by capturing its semantic composition with a bottom-up recursive neural network, and
- Utilization of relevant global structures for each constituent by propagating information with a top-down recursive neural network.

7.2 Future Work

The constituency-oriented approach is naturally suited to solve named entity recognition problem with nested or overlapped named entities, e.g. biological entities. This can be achieved by allowing the prediction collection phase described in Section 5 to consider all constituent nodes. One challenge of this research direction is likely to be parsing sentences in special domains.

doi:10.6342/NTU201704465

Besides nested predictions, a direct follow-up research direction is the development of an end-to-end method that unifies constituency graph generation and constituent classification in one model. This enables data-driven fine-tuning of parsing and graph generation according to the specific target task of constituent classification. The main challenge lies in back-propagating through parsing. In addition, restructuring algorithms designed with linguistic insights such as head-driven binarization might not be learned automatically.

In summary, the approach proposed in the thesis is able to classify linguistic units and hence identify text chunks of interest. NER is a prominent application of the approach. Furthermore, all the tasks about identifying linguistic units of interest might be better solved by the constituency-oriented approach than by traditional sequential labeling approaches.

doi:10.6342/NTU201704465

Bibliography

[1] D. Andor, C. Alberti, D. Weiss, A. Severyn, A. Presta, K. Ganchev, S. Petrov, and M. Collins. Globally normalized transition-based neural networks. *arXiv preprint arXiv:1603.06042*, 2016.

[2] N. Chinchor and P. Robinson. MUC-7 Named Entity Task Definition. In *Proceedings of the 7th Conference on Message Understanding*, volume 29, 1997.

[3] J. P. Chiu and E. Nichols. Named Entity Recognition with Bidirectional LSTM-CNNs. *Transactions of the Association for Computational Linguistics*, 4:357–370, 2016.

[4] M. Collins. *Head-Driven Statistical Models for Natural Language Parsing*. PhD thesis, University of Pennsylvania, 1999.

[5] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural Language Processing (Almost) from Scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537, 2011.

[6] G. Cybenko. Approximation by Superpositions of a Sigmoidal Function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, 1989.

[7] J. Daiber, M. Jakob, C. Hokamp, and P. N. Mendes. Improving Efficiency and Accuracy in Multilingual Entity Extraction. In *Proceedings of the 9th International Conference on Semantic Systems (I-Semantics)*, 2013.

doi:10.6342/NTU201704465

[8] C. dos Santos and V. Guimaraes. Boosting Named Entity Recognition with Neural Character Embeddings. In *Proceedings of the Fifth Named Entity Workshop, joint with 53rd ACL and the 7th IJCNLP*, pages 25–33, 2015.

[9] G. Durrett and D. Klein. A Joint Model for Entity Analysis: Coreference, Typing, and Linking. *Transactions of the Association for Computational Linguistics*, 2:477–490, 2014.

[10] J. R. Finkel and C. D. Manning. Joint Parsing and Named Entity Recognition. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 326–334. Association for Computational Linguistics, 2009.

[11] X. Glorot and Y. Bengio. Understanding the Difficulty of Training Deep Feedforward Neural Networks. In *International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

[12] R. Grishman and B. Sundheim. Message Understanding Conference-6: A Brief History. In *Coling*, volume 96, pages 466–471, 1996.

[13] E. Hovy, M. Marcus, M. Palmer, L. Ramshaw, and R. Weischedel. OntoNotes: The 90% Solution. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*, pages 57–60. Association for Computational Linguistics, 2006.

[14] O. Irsoy and C. Cardie. Bidirectional Recursive Neural Networks for Token-Level Labeling with Structure. In *NIPS Deep Learning Workshop*, 2013.

[15] Y. Kim, Y. Jernite, D. Sontag, and A. M. Rush. Character-Aware Neural Language Models. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[16] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980 [cs.LG]*, 2014.

doi:10.6342/NTU201704465

[17] M. Krallinger, F. Leitner, O. Rabal, M. Vazquez, J. Oyarzabal, and A. Valencia. Overview of the Chemical Compound and Drug Name Recognition (CHEMDNER) Task. In *BioCreative challenge evaluation workshop*, volume 2, page 2, 2013.

[18] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. Van Kleef, S. Auer, et al. DBpedia–A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia. *Semantic Web*, 6(2):167–195, 2015.

[19] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. RCV1: A New Benchmark Collection for Text Categorization Research. *Journal of machine learning research*, 5(Apr):361–397, 2004.

[20] G. Luo, X. Huang, C.-Y. Lin, and Z. Nie. Joint Named Entity Recognition and Disambiguation. In *Proc. EMNLP*, pages 879–888, 2015.

[21] E. Marsh and D. Perzanowski. MUC-7 Evaluation of IE Technology: Overview of Results. In *Proceedings of the seventh message understanding conference (MUC-7)*, volume 20, 1998.

[22] A. Passos, V. Kumar, and A. McCallum. Lexicon Infused Phrase Embeddings for Named Entity Resolution. In *Proceedings of the Eighteenth Conference on Computational Language Learning*, pages 78–86, 2014.

[23] J. Pennington, R. Socher, and C. D. Manning. Glove: Global Vectors for Word Representation. In *EMNLP*, volume 14, pages 1532–1543, 2014.

[24] S. Pradhan, A. Moschitti, N. Xue, H. T. Ng, A. Björkelund, O. Uryupina, Y. Zhang, and Z. Zhong. Towards Robust Linguistic Analysis using OntoNotes. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 143–152, 2013.

[25] L. Ratinov and D. Roth. Design Challenges and Misconceptions in Named Entity Recognition. In *Proceedings of the Thirteenth Conference on Computational Nat-*

doi:10.6342/NTU201704465

ural Language Learning, CoNLL '09, pages 147–155, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[26] R. Socher. *Recursive Deep Learning for Natural Language Processing and Computer Vision.* PhD thesis, Department of Computer Science, Stanford University, 2014.

[27] R. Socher, J. Bauer, C. D. Manning, and A. Y. Ng. Parsing with Compositional Vector Grammars. In *Proceedings of the ACL conference*, 2013.

[28] R. Socher, C. D. Manning, and A. Y. Ng. Learning Continuous Phrase Representations and Syntactic Parsing with Recursive Neural Networks. In *Proceedings of the NIPS-2010 Deep Learning and Unsupervised Feature Learning Workshop*, 2010.

[29] R. Socher, A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, 2013.

[30] K. S. Tai, R. Socher, and C. D. Manning. Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processingg*, pages 1556–1566, 2015.

[31] E. F. Tjong Kim Sang and F. De Meulder. Introduction to the CoNLL-2003 Shared Task: Language-independent Named Entity Recognition. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 142–147. Association for Computational Linguistics, 2003.

[32] R. Weischedel, M. Palmer, M. Marcus, E. Hovy, S. Pradhan, L. Ramshaw, N. Xue, A. Taylor, J. Kaufman, M. Franchini, et al. Ontonotes Release 5.0. *Linguistic Data Consortium, Philadelphia, PA*, 2013.

[33] S. Žitnik and M. Bajec. Token-and Constituent-based Linear-chain CRF with SVM

doi:10.6342/NTU201704465

for Named Entity Recognition. In *BioCreative Challenge Evaluation Workshop*, volume 2, page 144, 2013.

doi:10.6342/NTU201704465