INDUSTRIAL STRENGTH DEPENDENCY PARSING SYSTEM

by

Han He

THESIS

Presented to the Faculty of

The University of Houston-Clear Lake

In Partial Fulfillment

Of the Requirements

For the Degree

MASTER OF SCIENCE

in Computer Engineering

THE UNIVERSITY OF HOUSTON-CLEAR LAKE

MAY, 2018

INDUSTRIAL STRENGTH DEPENDENCY PARSING SYSTEM

by

Han He

APPROVED BY

Lei Wu, PhD, Chair

Xiaokun Yang, PhD, Committee Member

Liwen Shih, PhD, Committee Member

Jiang Lu, PhD, Committee Member

APPROVED/RECEIVED BY THE COLLEGE OF SCIENCE AND ENGINEERING:

Said Bettayeb, PhD, Associate Dean

Ju H. Kim, PhD, Dean

DEDICATION

I dedicate this dissertation to my friends and my parents. Without their encourage, understanding, and most of all love, the completion of this work would not have been possible.

ACKNOWLEDGMENTS

First and foremost, I am grateful to my major advisor, Dr. Lei Wu, for being friendly, caring, supportive, and helpful in numerous ways. Without his support, I could not have finished this work. He was very generous in sharing his experiences on computer science and engineering, academic life and beyond. He is not only my adviser, but also, a friend inspiring me for the rest of my life.

Next, I would like to thank the members of my committee, Dr. Xiaokun Yang, Dr. Liwen Shih, and Dr. Jiang Lu, for their support and suggestions in improving the quality of this dissertation. It is truly honored to have such fantastic and knowledgeable professors serving as my committee members.

I would also like to thank all the reviewers of those published papers which compose the majority of my thesis, for their assistance on work related to my research.

Furthermore, I would also like to acknowledge the research support provided from the Department of Computer Engineering at University of Houston-Clear Lake, and the dissertation year fellowship from the graduate school during my dissertation research.

Finally, I want to thank my family for their unconditional love, faith, and encouragement.

ABSTRACT

INDUSTRIAL STRENGTH DEPENDENCY PARSING SYSTEM

Han He
University of Houston-Clear Lake, 2018

Thesis Chair: Lei Wu, PhD

Dependency parsing is a useful task to help computer understand human language. By parsing the dependency grammar of a sentence automatically, dependency parser produces dependency-based syntactic representations which enhance performance of many language applications, such as machine translation, question answering and information extraction. Recently dependency parsing has attracted considerable interest from researchers and developers in the Natural Language Processing field, and many state-of-art works have achieved high accuracies. But not all of them are applicable for industry applications in terms of runtime speed and memory efficiency. We implemented and evaluated various dependency parsing algorithms, finding out the most practical algorithm in consideration of tradeoff between accuracy and runtime speed. The final achievement is a practically usable dependency parser, which can parse raw sentences to grammar trees. Our parser has been released as open source software and live demonstrated on `http://iparser.hankcs.com/`.

TABLE OF CONTENTS

Chapter Page

LIST OF TABLES

Table Page

LIST OF FIGURES

CHAPTER 1
INTRODUCTION TO DEPENDENCY PARSING

Dependency parsing is a task of automatically parsing dependency grammar for a sentence in natural language, which sustains advanced applications like machine translation, question answering etc. Starting with a brief introduction to dependency grammar and dependency parsing, this paper surveys influential and state-of-the-art works of the two major classes of approaches: transition based and graph based parsing. Instead of a through review, we mostly focus on fundamental concepts and current trends. To make a concrete study, we also analyzed state-of-the-art implementations of different approaches.

1.1 Introduction

Regarding automatic syntactic analysis of natural language, there are two families of grammar formalisms: phrase structure grammars and dependency grammars. They respectively correspond to constituent parsing and dependency parsing. Recently dependency parsing has become increasingly popular, mainly due to its flexibility and reliability for a wide range of different languages. Here we'll briefly introduce the dependency grammar as a background, then define the task of dependency parsing.

1.1.1 Dependency Grammar

Dependency grammar is a traditional concept dating back far to research work on linguistics, syntax and semantics by an ancient Sanskrit grammarian called Pāṇini in 6th to 4th century BCE. Although modern linguistic researchers regard the influential French linguist Lucien Tesnière as the founder of dependency theory, Pāṇini's devel-

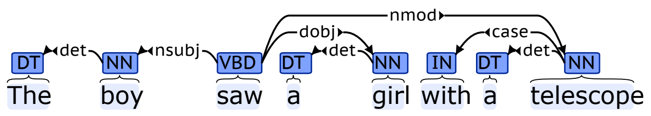


Figure 1.1: *High attachment dependency tree*

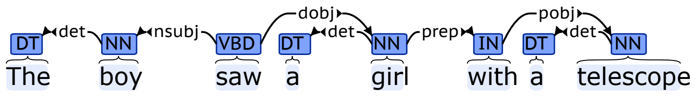


Figure 1.2: *Low attachment dependency tree*

opment of an approach to the syntax of natural languages is known as dependency grammar.

1.1.1.1 Basic Concepts

The basic assumption of dependency grammar is that language has a defined syntactic structure consisted of **words** and their **relations**. A dependency relation is a binary, asymmetrical links between a syntactically subordinate word, called the **dependent**, and another word on which it depends, called the **head** [47]. When every word is linked to its dependent by arrow with a label indicating the dependency type, a sentence is parsed into a **dependency tree**. For instance, given sentence ”The boy saw a girl with a telescope”, there are two plausible dependency trees illustrated in Figure 1.1 and Figure 1.2.

In dependency trees above, blue squares represent for part-of-speech tags listed in Table 1.1, labels on arrows are dependency types listed in Table 1.2. For example, the noun *boy* is a dependent of the verb *saw* with the dependency type *noun subject* (nsubj). The rest of the sentence cause a noticeable ambiguity called prepositional phrase (PP) attachment. Even based on lexical preferences, this PP attachment ambiguity can’t be resolved. As Hassanien et al. [33] explained, if high attachment is adopted, the PP (with a telescope) is attached to the verb (saw) which indicates

Tag	**Description**	**Tag**	**Description**
DT	Determiner	IN	Preposition
NN	Noun	VBD	Verb, past tense

Table 1.1: *Part-Of-Speech Tags*

Relation	**Description**
det	determiner
nsubj	nominal subject
prep	prepositional phrase
pobj	object of preposition
dobj	direct object
nmod	nominal modifier
case	case marking

Table 1.2: *Dependency Relation Types*

its relation to the verb (He used the telescope to see the girl). In the low attachment case, PP is attached to the nearest word (girl), which marks the coexistence of the PP with that word (He saw a girl who brings a telescope with her).

Corpora consisted of many parsed dependency trees are often called treebanks. Different treebanks defined heterogeneous part-of-speech tagsets and dependency relations. Our examples are instances of Penn Treebank part-of-speech tagset and Stanford Dependencies.

1.1.1.2 Projective Dependency Tree

A formal definition of dependency tree needs extra notations. Define a sentence S as a sequence of words $S = w_1 w_2 \cdots w_n$. Let $R = \{r_1, \cdots, r_m\}$ be a finite set of dependency relation types. Then a dependency tree is an directed acyclic graph:

$$G = (V, A)$$

where $V \subseteq \{w_0, w_1, \cdots, w_n\}$, $A \subseteq V \times R \times V$ and if arc $(w_i, r, w_j) \in A$ then $(w_i, r', w_j) \notin A$ for all $r \neq r'$. Specially, the node w_0 =ROOT is an artificial word simplifying both formal definitions and computational implementations. With its existence, every real word of the sentence have a syntactic head.

A projective dependency tree has a stricter limitation on arcs. Every word between arc (w_i, r, w_j) mustn't have dependent out of range (i, j). Although one language can have non-projective sentences, many recent dependency parsers only focus on projective dependency trees. In this paper, we concentrate on projective parsers too.

1.1.2 Dependency Parsing

Having introduced the basic concepts of dependency grammar, the definition of **dependency parsing** is to automatically analyze the dependency structure of a given input sentence. Given a sentence S consisting of words $w_0 w_1 w_2 \cdots w_n$, the parsing problem can be formally defined as mapping S to its dependency tree G.

Broadly speaking, there are two categories of parsing approaches, namely data-driven and rule-based (grammar-based). If an approach essentially uses machine learning techniques to learn patterns from dependency treebanks in order to parse new sentences, then it falls into the data-driven class. A rule-based approach typically relies on a formal grammar defining the rules a formal language should obey and apply that rule to new sentences. Recently dependency parsing is dominated by statistical methods on large treebanks. This paper only covers these data-driven solutions, or more precisely supervised methods, that is inputs in training set have been annotated with correct dependency structures.

In data-driven parsing, a dependency parsing model is denoted by

$$M = (\Gamma, \theta, h)$$

where Γ stands for constraints like projective restrictions, θ is a set of parameters belonging to that model, and h is a fixed parsing algorithm.

Transition	**Definition**	**Precondition**
Shift	$(\sigma, w_i\|\beta, A) \Rightarrow (\sigma\|w_i, \beta, A)$	
Left-Arc$_r$	$(\sigma\|w_i\|w_j, \beta, A) \Rightarrow (\sigma\|w_j, \beta, A \cup (w_j, r, w_i))$	$i \neq 0$
Right-Arc$_r$	$(\sigma\|w_i\|w_j, \beta, A) \Rightarrow (\sigma\|w_i, \beta, A \cup (w_i, r, w_j))$	

Table 1.3: *Arc-Standard transition system*

Transition	**Definition**	**Precondition**
Shift	$(\sigma, w_i\|\beta, A) \Rightarrow (\sigma\|w_i, \beta, A)$	
Left-Arc$_r$	$(\sigma\|w_i\|w_j, w_k\|\beta, A) \Rightarrow (\sigma\|w_i, w_k\|\beta, A \cup (w_k, r, w_j))$	$i \neq 0$
Right-Arc$_r$	$(\sigma\|w_i\|w_j, \beta, A) \Rightarrow (\sigma\|w_i, \beta, A \cup (w_i, r, w_j))$	

Table 1.4: *Arc-Hybrid transition system*

Statistical approaches can be divided into different classes, based on type of parsing model and algorithms. This paper focus on the two major classes, which are called transition-based and graph-based in literature.

These two classes differ in views about whether target dependency tree pre-exists or not. Transition-based methods assume no existence of tree at beginning. They start by defining a set of actions or a transition system for building a parse tree from scratch. Given input and current state including transition history, the learning problem is to train a model for predicting the next action to take. In this way, parsing problem is casted to construct the optimal transition sequence of the input sentence. Instead of constructing, graph-based methods choose the tree with highest score in a defined space of candidate dependency trees. They train a model to assign scores to the candidate dependency trees for a sentence, with the goal of letting target tree have the highest score. Then parsing problem becomes searching for the highest scoring dependency tree of the input sentence. These two classes of methods are treated in next two chapters in detail.

1.2 Transition-based Parsing

A transition-based parser starts with defining configurations (a set of states) and a set of possible actions for constructing a tree, called a transition system. A transition system often defines the constraints Γ. For instance, the arc-standard transition system ensures the output trees are projective.

After a transition system is fixed, dependency trees in training set are transformed into sequences of transition actions called oracle. Then a machine learning model θ is employed to recognize patterns between these transition sequences and input strings. Although a sequence model seems capable of such task, but usually a classifier is used instead. Because the interactions between adjacent actions are very limited. Modeling unrelated action interactions will damage the learning of a sequence model.

The parsing algorithms h for transition-based parsing are often greedy for fast runtime speed. Occasionally beam search algorithm is employed for slightly better performance.

1.2.1 Transition Systems

The terminology transition indicates that a transition system shares some similarities with a finite state automaton. By defining a finite set of configurations (states in FSA) and transitions (transition table in FSA), a transition system acts like a FSA, accepting an input string and performing a sequence of transitions from initial state to terminal state.

Given a set R of dependency relations, for a sentence $S = w_0 w_1 w_2 \cdots w_n$, a configuration is defined as a triple:

$$c = (\sigma, \beta, A)$$

where σ is a stack of words $w_i \in S$, β is a qunue of words $w_i \in S$ and A is a set of dependency arcs $(w_i, r, w_j) \subseteq V \times R \times V$.

A configuration is current state of the parsing process, representing an intermediate result of the input sentence. Concretely speaking the stack σ stores roots of parsed subtrees, while words in the queue β are the remaining input words.

Under this definition, the initial configuration is $([w_0]_\sigma, [w_1, \ldots, w_n]_\beta, \emptyset)$, which means all words are not parsed yet. A terminal configuration is $(\sigma, []_\beta, A)$ for any σ and A, which indicates that all words are parsed.

A transition is a partial function from configurations to configurations. The most widely used approach is stack-based transition systems implementing shift-reduce actions, where shift means move head word in β to σ, reduce means connect head word in β with top word in σ to build a tree. Yamada and Matsumoto [83] and Nivre [63] first introduced greedy transition-based parsing. Then other extensions are proposed to handle non-projective trees.

1.2.1.1 Arc-Standard

Arc-Standard [64] is the underlying system of many later extensions. It is consisted of actions shown in Table 1.3.

When each of the three actions is performed, top two words on stack σ or head word in queue β are the only elements to be operated, either connect top two words with an arc labeled as r, or simply push the later into the stack.

The Left-Arc$_r$ transition removes the top word w_i of the stack σ and attaches it as a modifier to the second word w_j in stack with label r, thus adding the arc (w_j, r, w_i). The Right-Arc$_r$ transition acts similar to Left-Arc$_r$ except that direction of arc is inverted, thus generating the arc (w_i, r, w_j). The Shift transition simply moves the first word w_i of the queue β to the stack σ.

1.2.1.2 Arc-Hybrid

Similar to the more popular arc-standard system, arc-hybrid transition system [48] is another implementation of shift-reduce actions. The difference between them is which two elements to be operated in left-arc action. Instead of operating the top two words in the stack, Arc-Hybrid system removes the first word on top of the stack σ and attaches it as a modifier to the head element in queue β with label r.

1.2.2 Oracle

An oracle is an algorithm which takes a sentence's gold parse tree as input, and predicts the sequence(s) of actions the parsing algorithm should follow in order to reproduce that gold tree, or at least the best reachable tree. Deterministic oracle is called static oracle whereas non-deterministic oracle is called dynamic oracle.

1.2.2.1 Static Oracle

Several transition sequences can produce the same gold tree, static oracles only map the gold tree to one of them. Thus they always produce one single static sequence of transitions supposed to be followed by the parser. The proposed transition sequence is not guaranteed to be the easiest or optimal one to learn.

Another limitation of static oracles is that, they are only defined as functions from one gold tree to one transition sequence, not as functions from configurations to transitions. When a parser makes mistakes, gold tree becomes unreachable, static oracles can't provide teaching signal of how to deal with such deviations. This leads to error-propagation.

1.2.2.2 Dynamic Oracle

Instead of being limited to a single static canonical transition sequence for producing a given gold tree, Goldberg and Nivre [30] introduced the dynamic oracle, which defines all transition sequences in a given configuration for producing a tree with minimum loss compared to the gold tree.

Training When configuration is a part of a gold derivation, dynamic oracle permits all valid transition sequences, otherwise it permits the most plausible ones for the best reachable tree.

Decoding The parser learnt to choose the special configurations from which the most gold arcs are still reachable. This preference can mitigate error-propagation while presuming the gold tree.

1.2.3 Feature Function

When we converted trees into configurations (as input) and valid transitions (as output), parsing task is then casted to the prediction of best transition given current configuration. The very first step towards this classification problem is to represent the configuration as a set of features which can be processed by a computer.

1.2.3.1 Traditional Feature Engineering

Traditionally, parsers rely on hand-crafted feature functions. These functions check whether current configuration satisfy some conditions then produces binary outputs like 1 for yes, 0 for no. For instance, ”distance between the head and the modifier words is 3 or not", "top word of the stack is 'he' or not", "rightmost child of the first word in queue is noun or not".

These elements being considered are well defined by a human expert, as well as the combinations of them. Zhang and Nivre [86] proposed an influential set of feature functions consisted of 20 elements and 72 feature templates, which are widely adopted in later works.

1.2.3.2 Feature Representation Learning

Recently, neural networks or deep learning brought automatically extracted, dense feature vectors to replace the sparse, binary indicator features.

Chen and Manning [12] is the first to embed those raw features into dense low-dimensional vector, and then feed those feature vectors into a two-layer perceptron, letting it potentially produce arbitrary feature combinations. They abandoned feature template completely. Then later works focusing on automatically feature combinations, but with other improvements including beam search are presented [80].

More than just tackling the effort in feature combinations, representation learning also helps feature engineering. Le and Zuidema [52] encoded tree node as two compositional representations built from both bottom-up and top-down. Dyer et al. [25] employed stack-LSTMs to encode the entire stack and buffer. Apart from these complex models, Kiperwasser and Goldberg [46] showed a simple Bi-LSTM can effectively capture enough feature information from each sentence token's sentential context, which will be covered in detail.

LSTM Long Short-term Memory Networks (LSTMs) are extensions of Recurrent Neural Networks (RNNs). They are designed to combat gradient vanishing issue by incorporating a memory-cell and are able to capture long-range dependencies.

Denote $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$ as a sequence of input vectors, the goal is to return another sequence $(\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_n)$ representing some information about the inputs at every time step. The following implementation is applied:

$$\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{W}_{xi}\mathbf{x}_t + \mathbf{W}_{hi}\mathbf{h}_{t-1} + \mathbf{W}_{ci}\mathbf{c}_{t-1} + \mathbf{b}_i) \\
\mathbf{c}_t &= (1 - \mathbf{i}_t) \odot \mathbf{c}_{t-1} + \\
&\qquad \mathbf{i}_t \odot \tanh(\mathbf{W}_{xc}\mathbf{x}_t + \mathbf{W}_{hc}\mathbf{h}_{t-1} + \mathbf{b}_c) \\
\mathbf{o}_t &= \sigma(\mathbf{W}_{xo}\mathbf{x}_t + \mathbf{W}_{ho}\mathbf{h}_{t-1} + \mathbf{W}_{co}\mathbf{c}_t + \mathbf{b}_o) \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t),
\end{aligned}$$

where σ is the element-wise sigmoid function.

Bi-LSTM Given a sentence $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$ containing n words represented as a d-dimensional vector, one LSTM can only produce a representation $\overrightarrow{\mathbf{h}_t}$ of the left context at every token t. To incorporate a representation of the right context $\overleftarrow{\mathbf{h}_t}$, a second LSTM which reads the same sequence in reverse is used. Pair of this forward and backward LSTM is called bidirectional LSTM(Bi-LSTM) [31] in literature. There are two directional LSTM producing two context representations of each word at time step t. By concatenating its left and right context representations, the final representation is produced as $\mathbf{h}_t = [\overrightarrow{\mathbf{h}_t}; \overleftarrow{\mathbf{h}_t}]$.

Bi-LSTM Feature Function As a simple and effective feature detector for sequential, Bi-LSTM has been applied to various tasks from chunking to neural machine translation. In dependency parsing, the feature function $\phi(c)$ is typically defined as concatenated Bi-LSTM vectors of several items on the stack and buffer. I.e., for a configuration $c = (\sigma|\ldots|s_1|s_0,\ \ b_0|b_1|\ldots|\beta,\ \ A)$ the feature extractor may be defined as:

$$\phi(c) = \mathbf{h}_{s_2} \circ \mathbf{h}_{s_1} \circ \mathbf{h}_{s_0} \circ \mathbf{h}_{b_0}$$

$$\mathbf{h}_i = \textsc{BiLstm}(x_{1:n}, i)$$

where $\circ$ denotes for vector concatenation operation.

1.2.4 Scoring Function

In transition based parsing, classification scoring function is borrowed when casted to classification task. When linear model is adopted, the scoring function will be:

$$\textsc{Score}_W(x, t) = (W \cdot x)[t]$$

where $x = \phi(c)$, W is the learnable model parameter, t is the transition being estimated. The linearity of single layer perceptron required the feature function $\phi(\cdot)$ to encode non-linearities via a complex feature template.

Recently, multilayer perceptron is commonly adopted to replace linear models, defined as:

$$\textsc{Score}_\theta(x, t) = MLP_\theta(x)[t]$$

where θ is a collection of trainable model parameters.

1.2.5 Training

The objective of training is similar to a common classification problem, thus to maximize the score of correct transitions above incorrect ones. The loss function is defined as:

$$\max_{t_p \in T \setminus G} MLP\big(\phi(c)\big)[t_p] - \max_{t_o \in G} MLP\big(\phi(c)\big)[t_o]$$

where T is the set of possible transitions and G is the set of gold transitions at the current stage.

1.2.6 Parsing Algoritm

In decoding phase, a greedy algorithm makes decision one configuration by one configuration, where as beam search algorithms keep track of the k most promising partial transition sequences after each transition step and search for the maximal sum of transitions by dynamic programing. For instance, a greedy parsing algorithm is presented in Algorithm 1 below.

Algorithm 1 Greedy transition-based parsing

```
1: Input: sentence $S = w_1, \ldots, x_w$, possible transitions $T = t_1, \ldots, t_n$, scoring
   function $\textsc{Score}_\theta(\cdot)$ with parameters $\theta$.
2: $c \leftarrow \textsc{Initial}(s)$
3: while not $\textsc{Terminal}(c)$ do
4:     $\hat{t} \leftarrow \arg\max_{t \in \text{Legal}(c)} \text{Score}_\theta\big(\phi(c), t\big)$
5:     $c \leftarrow \hat{t}(c)$
6: return $tree(c)$
```

1.3 Graph-based Parsing

A tree is essentially a directed acyclic graph, so it's naturally to apply standard graph algorithms to create dependency tree. In graph-based parsing, parser directly model substructures of a dependency tree, instead of indirect modeling over transition sequences to construct a tree. The target tree is hidden in a set of graphs consisted of all the words and relations, graph-based parser learns how to find it out.

	PTB		CTB	
Model	**UAS**	**LAS**	**UAS**	**LAS**
Transition				
Ballesteros et al. [2][1]	93.56	91.42	87.65	86.21
Andor et al. [1][2]	94.61	92.79	–	–
Kuncoro et al. [49][3]	**95.8**	**94.6**	–	–
Graph				
Kiperwasser and Goldberg [46][4]	93.9	91.9	87.6	86.1
Cheng et al. [16][5]	94.10	91.49	88.1	85.7
Dozat and Manning [22] [6]	95.74	94.08	**89.30**	**88.23**

Table 1.5: *Test-set parsing results of various state-of-the-art parsing systems on the English (PTB) and Chinese (CTB) datasets. Table taken from Dozat and Manning [22].*

Similar to transition-based model, a graph-based model $M = (\Gamma, \theta, h)$ consists of constraints Γ, model parameters θ and a deterministic parsing algorithm h. The parsing algorithm assigns a real value score to every dependency tree $G = (V, A) \in P_S$ for a sentence S:

$$\textsc{Score}(G) = \textsc{Score}(V, A) \in \mathbb{R}$$

where P_S is all the possible trees satisfying constraints Γ for sentence S. This score can be non-probabilistic values, conditional or joint probabilities.

Then this score for a whole tree G is assumed to factor through scores of subgraphs in G:

$$\textsc{Score}(G) = f\left(\psi_1, \psi_2, \ldots, \psi_q\right)$$

for all $\psi_i \in \psi_{\mathbf{G}}$, where $\psi_{\mathbf{G}}$ is the set of all subgraphs of G, f is some function over those subgraphs.

1.3.1 Arc-factored Models

The most commonly used parameterization function f is a summation over single dependency arcs (head, modifier in the arc and the arc's label):

$$f(\psi_1, \psi_2, \ldots, \psi_q) = \sum_{\psi_i \in \psi_{\mathbf{G}}} \textsc{Score}(\psi_i)$$

This makes arc-factored models follow the common structured prediction paradigm [78]:

$$\begin{aligned} \hat{G} &= \arg\max{}_{G \in P_s} \text{Score}(G) \\ &= \sum_{\psi_i \in \psi_{\mathbf{G}}} \text{Score}(\psi_i) \end{aligned}$$

In this way, score of whole graph is decomposed to the sum of arc scores in that graph.

1.3.2 Feature Function and Score Function

This two parts are mostly the same with transition parsers introduced in previous chapters.

Feature Function As there are no stacks and buffers, the core feature elements used are features related to the head word and the modifier word, typically their Bi-LSTM encodings [46].

Score Function Traditionally a linear model parameterized by a weight vector and feature vector is used by many decent works, then MLP took the charge since Pei et al. [66].

1.3.3 Training and Decoding

Training Similar to transition-based parsing, loss function is defined to maximize the score of gold parsing tree:

$$\max_{G' \neq G} \textsc{Score}(G') - \textsc{Score}(G)$$

where G is the gold parsing tree.

1.3.4 Decoding

Once local score for an arc is computed, arc-factored dependency parsing is equal to find the minimal spanning tree (MST) of a graph. To ensure projective constrains, Eisner [26] introduced an effective algorithm by breaking arc construction into smaller steps where a dynamic programming technique can be employed.

1.4 Implementations

In this chapter, we collect performance and implementations from several state-of-the-art works listed in Table 1.5, which can serve as a reading list. We also notice that Chinese dependency parsing is harder than English for both transition and graph based parsers.

1.5 Conclusion

We conduct a brief and incomplete survey of dependency parsing via two different approaches. Both of them shares similar feature extraction and scoring function, but differs a lot in parsing algorithms. We hope this paper could serve as a brief introduction to dependency parsing for beginners.

The rest of this dissertation is organized as follows. In Chapter 2, we propose a novel method for Chinese Word Segmentation, with consideration of components inside Chinese characters. In Chapter 3, we propose a joint learning framework to exploit heterogeneous corpora. In Chapter 4, we introduce our IParser, an industrial strength multilingual parser for human language processing, which is capable of parsing raw text to dependency grammar trees, with tokenizers and part-of-speech taggers built-in.

Finally, in Chapter 5, we conclude this dissertation and discuss possible future work.

[1]`https://github.com/clab/lstm-parser-dynamic`
[2]`https://github.com/tensorflow/models/tree/master/research/syntaxnet`
[3]`https://github.com/clab/rnng/tree/master/interpreting-rnng`
[4]`https://github.com/elikip/bist-parser`
[5]`https://github.com/hao-cheng/biattdp`
[6]`https://github.com/tdozat/Parser`

CHAPTER 2

DUAL LONG SHORT-TERM MEMORY NETWORKS FOR SUB-CHARACTER REPRESENTATION LEARNING

Characters have commonly been regarded as the minimal processing unit in Natural Language Processing (NLP). But many non-latin languages have hieroglyphic writing systems, involving a big alphabet with thousands or millions of characters. Each character is composed of even smaller parts, which are often ignored by the previous work. In this paper, we propose a novel architecture employing two stacked Long Short-Term Memory Networks (LSTMs) to learn sub-character level representation and capture deeper level of semantic meanings. To build a concrete study and substantiate the efficiency of our neural architecture, we take Chinese Word Segmentation as a research case example. Among those languages, Chinese is a typical case, for which every character contains several components called radicals. Our networks employ a shared radical level embedding to solve both Simplified and Traditional Chinese Word Segmentation, without extra Traditional to Simplified Chinese conversion, in such a highly end-to-end way the word segmentation can be significantly simplified compared to the previous work. Radical level embeddings can also capture deeper semantic meaning below character level and improve the system performance of learning. By tying radical and character embeddings together, the parameter count is reduced whereas semantic knowledge is shared and transferred between two levels, boosting the performance largely. On 3 out of 4 Bakeoff 2005 datasets, our method surpassed state-of-the-art results by up to 0.4%. Our results are reproducible, source codes and corpora are available on GitHub[1]

[1]`https://github.com/hankcs/sub-character-cws`

SC	Sem.	Pho.	TC	Sem.	Pho.
鲤	鱼	里	鯉	魚	里
鲢	鱼	连	鰱	魚	連
河	水	可	河	水	可
沟	水	勾	溝	水	冓
捞	手	劳	撈	手	勞
捡	手	佥	撿	手	僉

Table 2.1: *Illustration of semantic component (Sem.) and phonetic component (Pho.) in Simplified Chinese (SC) and Traditional Chinese (TC).*

2.1 Introduction

Unlike English, the alphabet in many non-latin languages is often big and complex. In those hieroglyphic writing systems, every character can be decomposed into smaller parts or sub-characters, and each part has special meanings. But existing methods often follow common processing steps in latin flavor [57, 58, 4, 45, 69], and treat character as the minimal processing unit, leading to a neglecting of information inside non-latin characters. Early work exploiting sub-character information usually treat it as a separate level from character [77, 53, 72, 21], ignoring the language phenomenon that some of those sub-characters themselves are often used as normal characters. From this phenomenon, we gained a new motivation to design a novel neural network architecture for learning character and sub-character representation jointly.

In linguists' view, Chinese writing system is such a highly hieroglyphic language, and it has a long history of character compositionality. Every Chinese character has several radicals ("部首"in Chinese), which serves as semantic component for encoding meaning, or phonetic component for representing pronouciation. For instance, we listed radicals of several Simplified and Traditional Chinese characters in Table 2.1. Chinese characters with same semantic component are closely correlated in semantic. As shown above, carp (鲤) and silverfish (鲢) are both fish (鱼). River (河) and gully

(沟) are all filled with water (水). To catch (捞) or to pick up (捡) a fish, one needs to use hands (手). To exploit those semantic meanings under character embedding level, radical embedding emerged since 2014 [77, 72, 59, 21]. These early work treated sub-character and character as two separate levels, omitting that they can actually be unified as single minimal processing unit in language model. Instead of ignoring linguistic knowledge, we respect the divergence of human language, and propose a novel joint learning framework for both character and sub-character representations.

To verify the efficiency of our jointly learnt representations, we conducted extensive experiments on the Chinese Word Segmentation (CWS) task. As those languages often don't have explicit delimiters between words, making it hard to perform later NLP tasks like Information Retrieval or Question Answering. Chinese language is such a typical non-segmented language, which means unlike English language having spaces between every word, Chinese has no explicit word delimiters. Therefore, Chinese Word Segmentation is a preliminary pre-processing step for later Chinese language process tasks. Recently with the rapid rise of deep learning, neural word segmentation approaches arose to reduce efforts in feature engineering [90, 18, 65, 14, 6, 7].

In this paper, we propose a novel model to dive deeper into character embeddings. In our framework, Simplified Chinese and Traditional Chinese corpora are unified via radical embedding, growing an end-to-end model. Every character is converted to a sequence of radicals with its original form. Character embeddings and radical embeddings are pretrained jointly in Bojanowski et al. [4]'s subword aware method. Finally, we conducted various experiments on corpora from SIGHAN bakeoff 2005. Results showed that our jointly learnt character embedding outperforms conventional character embedding training methods. Our models can improve performance by transfer learning between characters and radicals. The final scores surpassed previous work, and 3 out of 4 even surpassed previous preprocessing-heavy state-of-the-art learning work.

More specifically, the contributions of this paper could be summarized as:

- Explored a novel sub-character aware neural architecture and unified character and sub-character as one same level embedding.
- Released the first full Chinese character-radical conversion corpus along with pre-trained embeddings, which can be easily applied on other NLP tasks. Our codes and corpora are freely available for the public.

2.2 Related Work

In this section, we review the previous work from 2 directions – radical embedding and Chinese Word Segmentation.

2.2.1 Radical Embedding

To leverage the semantic meaning inside Chinese characters, Sun et al.[77] inaugurated radical information to enrich character embedding via softmax classification layer. In similar way, Li et al.[53] proposed charCBOW model taking concatenation of the character-level and component-level context embeddings as input. Making networks deeper, Shi et al.[72] proposed a deep CNN on top of radical embedding pre-trained via CBOW. Instead of utilizing CNNs, following Lample et al.[51], Dong et al.[21] used two level LSTMs taking character embedding and radical embedding as input respectively.

Our work is closely related to Dong et al.[21], but there are two major differences. In pre-training phase, their character embeddings were pre-trained separately, by utilizing conventional word2vec package, and the radical embeddings are randomly initialized. While we considered radical units as sub-characters (parts of one character) and trained the two level embeddings jointly, following Bojanowski et al. [4]'s approach. In training and testing phases, our two-level embeddings are tied up and unified as the sole minimal input unit of Chinese language.

2.2.2 Chinese Word Segmentation

Chinese Word Segmentation has been a well-known NLP task for decades[39]. After pioneer Xue et al.[82] transformed CWS into a character-based tagging problem, Peng et al. [67] adopted CRF as the sequence labeling model and showed its effectiveness. Following these pioneers, later sequence labeling based work [79, 88, 89, 76] was proposed. Recent neural models [90, 70, 65, 14, 21, 15] also followed this sequence labeling fashion.

Our model is based on Bi-LSTM with CRF as top layer. Unlike previous approaches, the inputs to our model are both character and radical embeddings. Furthermore, we explored which embedding level is more tailored for Chinese language, either using both embeddings together, or even tying them up.

2.3 Joint Learning for Character Embedding and Radical Embedding

Previous work treated character and radical as two different levels, used them separately or used one to enhance the other. Although radicals are components of a character (belonging to a lower level), they can actually be learnt jointly. It is linguistically more reasonable to put radical embeddings and character embeddings in exactly the same vector space. We propose to train character vector representation being aware of its internal structure of radicals.

2.3.1 Character Decomposition

Every character can be decomposed into a list of radicals or components. To maintain character information in radical list, we simply add the raw form of character to its radical list. Taking the linguistic knowledge that semantic component contains richest meaning of one character into consideration, we append the semantic component to the end of its radical list, hence to make the semantic component appear more than once.

Formally, denote c as a character, r as a radical, $\mathcal{L}_c = [r_1, r_2 \cdots r_n]$ as the original radical list of c. Let $r_s \in \mathcal{L}_c$ be the semantic component of c. Our decomposition of c will be:

$$\mathcal{R}_c = [c, r_1, r_2 \cdots r_n, r_s] \tag{2.1}$$

2.3.2 General Continuous Skip-Gram (SG) Model

Take a brief review of the continuous skip-gram model introduced by Mikolov et al.[59], applied in character representation learning.

Given an alphabet, target is to learn a vectorial representation $\mathbf{v}_c$ for each character c. Let $c_1, ..., c_T$ be a large-scale corpus represented as a sequence of characters, the objective function of the skipgram model is to maximize the log-likelihood of correct prediction. The probability of a context character c_y given c_x is computed by a scoring function s which maps character and context to scores in $\mathbb{R}$.

The general SG model ignores the radical structure of characters, we propose a different scoring function s, in order to capture radical information.

Let all radicals form an alphabet of size R. Given a character c and the radical list $\mathcal{R}_c \subset \{1, \dots, R\}$ of c, a vector representation $\mathbf{z}_r$ is associated to each radical r. Then a character is represented by the sum of the vector representations of its radicals. Thus the new scoring function will be:

$$s(c_x, c_y) = \sum_{r \in \mathcal{R}_{c_x}} \mathbf{z}_r^\top \mathbf{v}_{c_y}. \tag{2.2}$$

This simple model allows learning the representations of characters and radicals jointly.

2.4 Radical Aware Neural Architectures for General Chinese Word Segmentation

Once character and radical representations are learnt, one evaluation metric is how much it improves a NLP task. We choose the Chinese Word Segmentation task as a standard benchmark to examine their efficiency. One prevailing approach to CWS is casting it to character based sequence tagging problem, where our representations can be applied. A commonly used tagging set is $\mathcal{T} = \{B, M, E, S\}$, representing the **b**egin, **m**iddle, **e**nd of a word, or **s**ingle character forming a word.

Given a sequence $\mathbf{X}$ consisted of n features as $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n)$, the goal of sequence tagging based CWS is to find the most possible tags $\mathbf{Y}^* = \{\mathbf{y}_1^*, \dots, \mathbf{y}_n^*\}$:

$$\mathbf{Y}^* = \arg\max_{\mathbf{Y} \in \mathcal{T}^n} p(\mathbf{Y}|\mathbf{X}), \tag{2.3}$$

where $\mathcal{T} = \{B, M, E, S\}$.

Since tagging set restricts the order of adjacent tags, we model them jointly using a conditional random field, mostly following the architecture proposed by Lample et al.[51], via stacking two LSTMs with a CRF layer on top of them.

2.4.1 Radical LSTM Layer: Character Composition from Radicals

In this section, we'll review RNN with Bi-LSTM extension briefly, before introducing our character composition network.

LSTM Long Short-Term Memory Networks (LSTMs) [38] are extensions of Recurrent Neural Networks (RNNs). They are designed to combat gradient vanishing issue via incorporating a memory-cell which enables long-range dependencies capturing.

Bi-LSTM One LSTM can only produce the representation $\overrightarrow{\mathbf{h}_t}$ of the left context at every character t. To incorporate a representation of the right context $\overleftarrow{\mathbf{h}_t}$, a second LSTM which reads the same sequence in reverse order is used. Pair of this forward and backward LSTM is called bidirectional LSTM (Bi-LSTM) [31] in literature. By concatenating its left and right context representations, the final representation is produced as $\mathbf{h}_t = [\overrightarrow{\mathbf{h}_t}; \overleftarrow{\mathbf{h}_t}]$.

We apply a Bi-LSTM to compose character embeddings from radical embeddings in both directions. The raw character is inserted as the first radical, and the semantic component is appended as the last radical. The motivation behind this trick is to make use of LSTM's bias phenomena. In practice, LSTMs usually tend to be biased towards the most recent inputs of the sequence, thus the first one or last one depends on its direction.

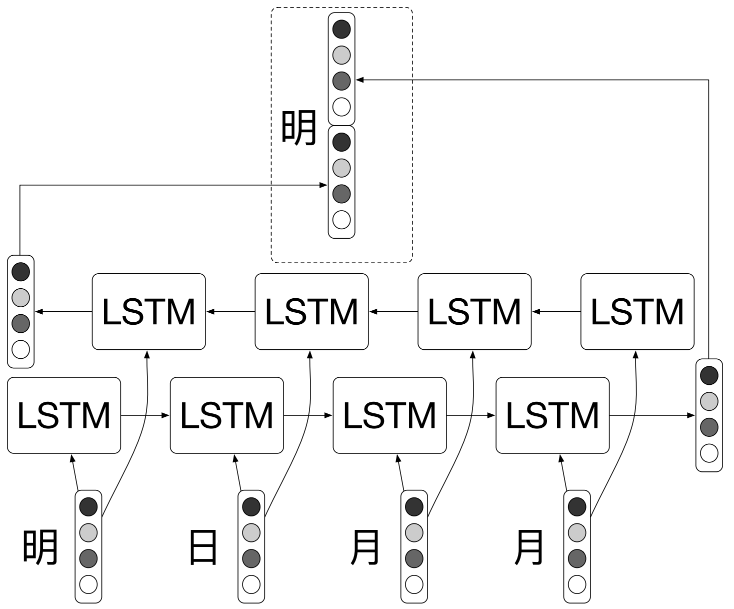


Figure 2.1: *Radical LSTM Layer – composition of character representation from radicals*

As illustrated in Figure 2.1, the character 明 (bright) has the radical list of 日 (sun) and 月 (moon) with its raw form and duplicated semantic radical. Its compositional representation $\mathbf{h}_i^r \in \mathbb{R}^{2k}$ is agglomerated via a Bi-LSTM from these radical embeddings, where k is the dimension of radical embeddings.

2.4.2 Character Bi-LSTM Layer: Context Capturing

Once compositional character representation $\mathbf{h}_i^r$ is synthesized, the contextual representation $\mathbf{h}_t^c \in \mathbb{R}^{2d}$ at every character t in input sentence can be agglomerated by a second Bi-LSTM. The dimension d is a flexible hyper-parameter, which will be explored in later experiments.

Our architecture for contextual feature capturing is shown in Figure 2.2. This contextual feature vector contains the meaning of a character, its radicals and its context.

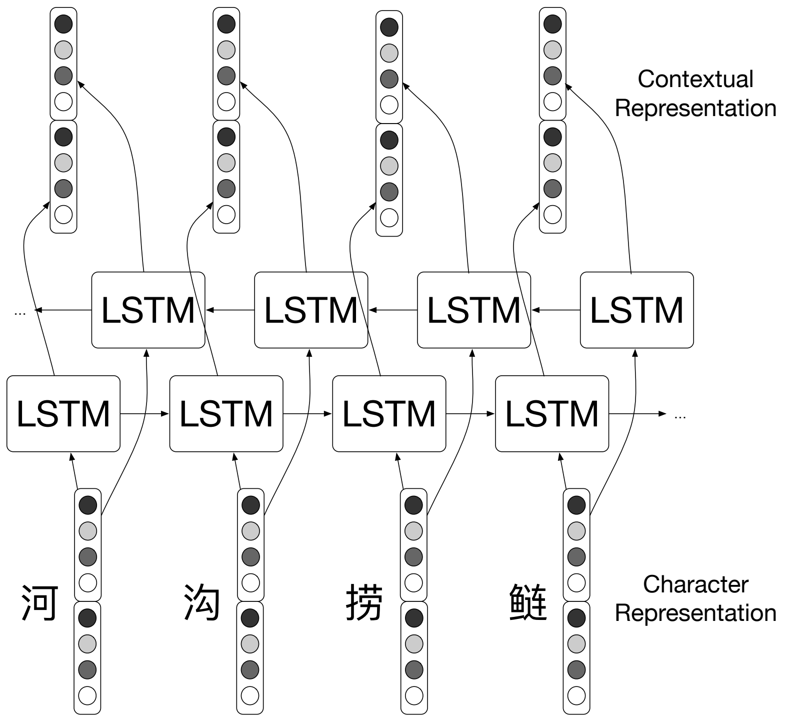


Figure 2.2: *Character LSTM Layer – capture contextual representation*

2.4.3 CRF Layer: Tagging Inference

We employed a Conditional Random Fields(CRF) [50] layer as the inference layer. As first order linear chain CRFs only model bigram interactions between output tags, so the maximum of a posteriori sequence $\mathbf{Y}^*$ in Eq. 4.1 can be computed using dynamic programming, both in training and decoding phase. The training goal is to maximize the log-probability of the gold tag sequence.

2.5 Experiments

We conducted various experiments to verify the following questions:

1. Does radical embedding enhance character embedding in pre-training phase?

2. Whether radical embedding helps character embedding in training phase and test phase (by using character embedding solely or using them both)?
3. Can radical embedding replace character embedding (by using radical embedding only)?
4. Should we tie up two level embeddings?

2.5.1 Datasets

To explore these questions, we experimented on the 4 prevalent CWS benchmark datasets from SIGHAN2005 [27]. Following conventions, the last 10% sentences of training set are used as development set.

2.5.2 Radical Decomposition

We obtained radical lists of character from the *online Xinhua Dictionary*[2], which are included in our open-source project.

2.5.3 Pre-training

Previous work have shown that pre-trained embeddings on large unlabeled corpus can improve performance. It usually involves lots of efforts to preprocess those corpus. Here we presented a novel solution.

The corpus used is Chinese Wikipedia of July 2017. Unlike most approaches, we don't perform Traditional Chinese to Simplified Chinese conversion. Our radical decomposition is sufficient of associate character to its similar variants. Not only

[2]`http://tool.httpcn.com/Zi/`

Models	PKU	MSR	CityU	AS
Tseng et al. [79]	95.0	96.4	-	-
Zhang and Clark [85]	95.0	96.4	-	-
Sun et al. [75]	95.2	97.3	-	-
Sun et al. [76]	95.4	**97.4**	-	-
Pei et al. [65]	95.2	97.2	-	-
Chen et al. [15]	94.3	96.0	95.6	94.8
Cai et al. [7]$^\diamondsuit$	**95.8**	97.1	95.6	95.3
baseline	94.6	96.0	94.7	94.8
+subchar	95.0	96.0	94.9	94.9
+radical	94.6	96.7	95.3	95.2
+radical -char	94.4	96.5	95.0	95.1
+radical +tie	94.8	96.8	95.3	95.1
+radical +tie +bigram	95.3	**97.4**	**95.9**	**95.7**

Table 2.2: *Comparison with previous state-of-the-art models of results on all four Bakeoff-2005 datasets.*

traditional-simplified character pairs, those with similar radical decompositions will also share similar vectorial representations.

Further, instead of the commonly used word2vec [58], we utilized fastText[3] [4] to train character embeddings and radical embeddings jointly. We applied SG model, 100 dimension, and set both maximum and minimal n-gram length to 1, as the radical takes only one token.

2.5.4 Final Results on SIGHAN bakeoff 2005

Our baseline model is Bi-LSTM-CRF trained on each datasets only with pre-trained character embedding (the conventional word2vec), no sub-character enhancement, no radical embeddings. Then we improved it with sub-character information, adding radical embeddings, tying two level embeddings up. The final results are shown in Table 3.3.

[3]`https://github.com/facebookresearch/fastText` With tiny modification to output n-gram vectors.

All experiments are conducted with standard Bakeoff scoring program[4] calculating precision, recall, and F_1-score. Note that results with $\diamondsuit$ expurgated long words in test set.

2.5.5 Model Analysis

Sub-character information enhances character embeddings. Previous work showed pre-trained character embeddings can improve performance. Our experiment showed with sub-character information (+subchar), performance can be further improved compared to no sub-character enhancement (baseline). By simply replacing the conventional word2vec embeddings to radical aware embeddings, the score can benefit an improvement as much as 0.4%.

Radical embeddings collaborate well with character embeddings. By building compositional embeddings from radical level (+radical), performance increased by up to 0.7% in comparison with model (baseline) on MSR dataset. But we also notice that: 1) On small dataset such as PKU, radical embeddings cause tiny performance drop. 2) With the additional bigram feature, performance can be further increased as much as 0.6%.

Radical embeddings can't fully replace character embeddings. Without character embeddings but use radical embeddings solely (+radical -char), performance drops a little (0.1% to 0.3%) compared to the model with character embeddings (+radical).

Tying two level embeddings up is a good idea. By tying radical embeddings and character embeddings together (+radical +tie), the raw feature is unified into the same vector space, knowledge is transferred between two levels, and performance is boosted up to 0.2%.

[4]`http://www.sighan.org/bakeoff2003/score` This script rounds a score to one digit.

2.6 Conclusions and Future Work

In this paper, we proposed a novel neural network architecture with dedicated pre-training techniques to learn character and sub-character representations jointly. As an concrete application example, we unified Simplified and Traditional Chinese characters through sub-character or radical embeddings. We have utilized a practical way to train radical and character embeddings jointly. Our experiments showed that sub-character information can enhance character representations for a pictographic language like Chinese. By using both level embeddings and tying them up, our model has gained the most benefit and surpassed previous single criterial CWS systems on 3 datasets.

Our radical embeddings framework can be applied to extensive NLP tasks like POS-tagging and Named Entity Recognition (NER) for various hieroglyphic languages. These tasks will benefit from deeper level of semantic representations encoded with more linguistic knowledge.

CHAPTER 3
EFFECTIVE NEURAL SOLUTION FOR MULTI-CRITERIA WORD SEGMENTATION

We present a simple yet elegant solution to train a single joint model on multi-criteria corpora for Chinese Word Segmentation (CWS). Our novel design requires no private layers in model architecture, instead, introduces two artificial tokens at the beginning and ending of input sentence to specify the required target criteria. The rest of the model including Long Short-Term Memory (LSTM) layer and Conditional Random Fields (CRFs) layer remains unchanged and is shared across all datasets, keeping the size of parameter collection minimal and constant. On Bakeoff 2005 and Bakeoff 2008 datasets, our innovative design has surpassed both single-criterion and multi-criteria state-of-the-art learning results. To the best knowledge, our design is the first one that has achieved the latest high performance on such large scale datasets. Source codes and corpora of this paper are available on GitHub[1].

3.1 Introduction

Unlike English language with space between every word, Chinese language has no explicit word delimiters. Therefore, Chinese Word Segmentation (CWS) is a preliminary pre-processing step for Chinese language processing tasks. Following Xue [82], most approaches consider this task as a sequence tagging task, and solve it with supervised learning models such as Maximum Entropy (ME) [43] and Conditional Random Fields (CRFs) [50, 67]. These early models require heavy handcrafted feature engineering within a fixed size window.

[1]`https://github.com/hankcs/multi-criteria-cws`

With the rapid development of deep learning, neural network word segmentation approach arose to reduce efforts in feature engineering [90, 18, 65, 14, 6, 7]. Zheng et al. [90] replaced raw character with its embedding as input, adapted the sliding-window based sequence labeling [18]. Pei et al. [65] extended Zheng et al. [90]'s work by exploiting tag embedding and bigram features. Chen et al. [14] employed LSTM to capture long-distance preceding context. Noteworthily, a novel word-based approach [6, 7] was proposed to model candidate segmented results directly. Despite the outstanding runtime performance, their solution required the max word length L to be a fixed hyper-parameter and replaced those words that longer than L into a unique character. Thus their performance relies on an expurgation of long words, which is not practical.

Novel algorithms and deep models are not omnipotent. Large-scale corpus is also important for an accurate CWS system. Although there are many segmentation corpora, these datasets are annotated in different criteria, making it hard to fully exploit these corpora, which are shown in Table 3.1.

Corpora	Li	Le	reaches	Benz	Inc
pku	李	乐	到达	奔驰	公司
msr	李乐		到达	奔驰公司	
as	李樂		到達	賓士	公司
cityu	李樂		到達	平治	公司

Table 3.1: *Illustration of different segmentation criteria of SIGHAN bakeoff 2005.*

Recently, Chen et al. [15] designed an adversarial multi-criteria learning framework for CWS. However, their models have several complex architectures, and are not comparable with the state-of-the-art results.

In this paper, we propose a smoothly jointed multi-criteria learning solution for CWS by adding two artificial tokens at the beginning and ending of input sentence to specify the required target criteria. We have conducted various experiments on 8 segmentation criteria corpora from SIGHAN Bakeoff 2005 and 2008. Our models improve performance by transferring learning on heterogeneous corpora. The final scores have surpassed previous multi-criteria learning, 2 out of 4 even have surpassed previous preprocessing-heavy state-of-the-art single-criterion learning results.

The contributions of this paper could be summarized as:

- Proposed an simple yet elegant solution to perform multi-criteria learning on multiple heterogeneous segmentation criteria corpora;
- 2 out of 4 datasets have surpassed the state-of-the-art scores on Bakeoff 2005;
- Extensive experiments on up to 8 datasets have shown that our novel solution has significantly improved the performance.

3.2 Related Work

In this section, we review the previous works from 2 directions, which are Chinese Word Segmentation and multi-task learning.

3.2.1 Chinese Word Segmentation

Chinese Word Segmentation has been a well-studied problem for decades [39]. After pioneer Xue [82] transformed CWS into a character-based tagging problem, Peng et al. [67] adopted CRF as the sequence labeling model and showed its effectiveness. Following these pioneers, later sequence labeling based works [79, 88, 89, 76] were proposed. Recent neural models [90, 65, 14, 21, 15] also followed this sequence labeling fashion.

3.2.2 Multi-Task Learning

Compared to single-task learning, multi-task learning is relatively harder due to the divergence between tasks and heterogeneous annotation datasets. Recent works have started to explore joint learning on Chinese word segmentation or part-of-speech tagging. Jiang et al. [41] stacked two classifiers together. The later one used the former's prediction as additional features. Sun and Wan [74] proposed a structure-based stacking model in which one tagger was designed to refine another tagger's prediction. These early models lacked a unified loss function and suffered from error propagation.

Qiu et al. [71] proposed to learn a mapping function between heterogeneous corpora. Li et al. [54], Chao et al. [8] proposed and utilized coupled sequence labeling model which can directly learn and infer two heterogeneous annotations simultaneously. These works mainly focused on exploiting relationships between different tagging sets, but not shared features.

Chen et al. [15] designed a complex framework involving sharing layers with Generative Adversarial Nets (GANs) to extract the criteria-invariant features and dataset related private layers to detect criteria-related features. This research work didn't show great advantage over previous state-of-the-art single-criterion learning scores.

Our solution is greatly motivated by Google's Multilingual Neural Machine Translation System, for which Johnson et al. [44] proposed an extremely simple solution without any complex architectures or private layers. They added an artificial token corresponding to parallel corpora and train them jointly, which inspired our design.

3.3 Neural Architectures for Chinese Word Segmentation

A prevailing approach to Chinese Word Segmentation is casting it to character based sequence tagging problem [82, 76]. One commonly used tagging set is $\mathcal{T} = \{\mathrm{B}, \mathrm{M}, \mathrm{E}, \mathrm{S}\}$, representing the **b**egin, **m**iddle, **e**nd of a word, or single character forming a word. Given a sequence $\mathbf{X}$ with n characters as $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$, sequence tagging based CWS is to find the most possible tags $\mathbf{Y}^* = \{\mathbf{y}_1^*, \ldots, \mathbf{y}_n^*\}$:

$$\mathbf{Y}^* = \underset{\mathbf{Y} \in \mathcal{T}^n}{\arg\max}\, p(\mathbf{Y}|\mathbf{X}), \tag{3.1}$$

We model them jointly using a conditional random field, mostly following the architecture proposed by Lample et al. [51], via stacking Long Short-Term Memory Networks (LSTMs) [38] with a CRFs layer on top of them.

We'll introduce our neural framework bottom-up. The bottom layer is a character Bi-LSTM (bidirectional Long Short-Term Memory Network) [31] taking character embeddings as input, outputs each character's contextual feature representation:

$$\mathbf{h}_t = \text{Bi-LSTM}(\mathbf{X}, t) \tag{3.2}$$

After a contextual representation $\mathbf{h}_t$ is generated, it will be decoded to make a final segmentation decision. We employed a Conditional Random Fields (CRF) [50] layer as the inference layer.

First of all, a linear score function $s(\mathbf{X}, t) \in \mathbb{R}^{|\mathcal{T}|}$ is used to assign a local score for each tag on t-th character:

$$s(\mathbf{X}, t) = \mathbf{W}_s^\top \mathbf{f}_t + \mathbf{b}_s \tag{3.3}$$

where $\mathbf{f}_t = [\mathbf{h}_t; \mathbf{e}_t]$ is the concatenation of Bi-LSTM hidden state and bigram feature embedding $\mathbf{e}_t$, $\mathbf{W}_s \in \mathbb{R}^{d_f \times |\mathcal{T}|}$ and $\mathbf{b}_s \in \mathbb{R}^{|\mathcal{T}|}$ are trainable parameters.

Then, for a sequence of predictions:

$$\mathbf{Y} = (y_1, y_2, \ldots, y_n) \tag{3.4}$$

first order linear chain CRFs employed a Markov chain to define its global score as:

$$s(\mathbf{X}, \mathbf{Y}) = \sum_{i=0}^{n} A_{y_i, y_{i+1}} + \sum_{i=1}^{n} P_{i, y_i} \tag{3.5}$$

where $\mathbf{A}$ is a transition matrix such that $A_{i,j}$ represents the score of a transition from the tag y_i to tag y_j. y_0 and y_n are the *start* and *end* tags of a sentence, that are added to the tagset additionaly. $\mathbf{A}$ is therefore a square matrix of size $4 + 2$.

Finally, this global score is normalized to a probability in Equation (4.1) via a softmax over all possible tag sequences:

$$p(\mathbf{Y}|\mathbf{X}) = \frac{e^{s(\mathbf{X},\mathbf{Y})}}{\sum_{\widetilde{\mathbf{Y}} \in \mathbf{Y_X}} e^{s(\mathbf{X},\widetilde{\mathbf{Y}})}} \tag{3.6}$$

In decoding phase, first order linear chain CRFs only model bigram interactions between output tags, so the maximum of a posteriori sequence $\mathbf{Y}^*$ in Eq. 4.1 can be computed using dynamic programming.

3.4 Elegant Solution for Multi-Criteria Chinese Word Segmentation

For closely related multiple task learning like multilingual translation system, Johnson et al. [44] proposed a simple and practical solution. It only needs to add an artificial token at the beginning of the input sentence to specify the required target language, no need to design complex private encoder-decoder structures.

We follow their spirit and add two artificial tokens at the beginning and ending of input sentence to specify the required target criteria. For instance, sentences in SIGHAN Bakeoff 2005 will be designed to have the following form:

These artificial tokens specify which dataset the sentence comes from. They are treated as normal tokens, or more specifically, a normal character. With their help,

Corpora	Li Le reaches Benz Inc
PKU	<pku> 李乐到达奔驰公司 </pku>
MSR	<msr> 李乐到达奔驰公司 </msr>
AS	<as> 李樂到達賓士公司 </as>
CityU	<cityu> 李樂到達平治公司 </cityu>

Table 3.2: *Illustration of adding artificial tokens into 4 datasets on SIGHAN Bakeoff 2005. To be fair, these <dataset> and </dataset> tokens will be removed when computing scores.*

instances from different datasets can be seamlessly put together and jointly trained, without extra efforts. These two special tokens are designed to carry criteria related information across long dependencies, affecting the context representation of every character, and finally to produce segmentation decisions matching target criteria. At test time, those tokens are used to specify the required segmentation criteria. Again, they won't be taken into account when computing performance scores.

3.5 Training

The training procedure is to maximize the log-probability of the gold tag sequence:

$$\begin{aligned}\log(p(\mathbf{Y}|\mathbf{X})) = &\ \text{score}(\mathbf{X}, \mathbf{Y}) \\ &- \underset{\widetilde{\mathbf{Y}} \in \mathbf{Y}_{\mathbf{X}}}{\text{logadd}}\ \text{score}(\mathbf{X}, \widetilde{\mathbf{Y}}),\end{aligned} \tag{3.7}$$

where $\mathbf{Y}_{\mathbf{X}}$ represents all possible tag sequences for a sentence $\mathbf{X}$.

3.6 Experiments

We conducted various experiments to verify the following questions:

1. Is our multi-criteria solution capable of learning heterogeneous datasets?

2. Can our solution be applied to large-scale corpus groups consisting of tiny and informal texts?
3. More data, better performance?

Our implementation is based on Dynet [62], a dynamic neural net framework for deep learning. Additionally, we implement the CRF layer in Python, and integrated the official score script to verify our scores.

3.6.1 Datasets

To explore the first question, we have experimented on the 4 prevalent CWS datasets from SIGHAN2005 [27] as these datasets are commonly used by previous state-of-the-art research works. To challenge question 2 and 3, we applied our solution on SIGHAN2008 datasets [61], which are used to compare our approach with other state-of-the-art multi-criteria learning works under a larger scale.

All datasets are preprocessed by replacing the continuous English characters and digits with a unique token. For training and development sets, lines are split into shorter sentences or clauses by punctuations, in order to make faster batch.

Specially, the Traditional Chinese corpora CityU, AS and CKIP are converted to Simplified Chinese using the popular Chinese NLP tool HanLP[2].

3.6.2 Results on SIGHAN bakeoff 2005

Our baseline model is Bi-LSTM-CRFs trained on each datasets separately. Then we improved it with multi-criteria learning. In naive treatment (+naive), artificial tokens

[2]`https://github.com/hankcs/HanLP`

Models	PKU	MSR	CityU	AS
Tseng et al. [79]	95.0	96.4	-	-
Zhang and Clark [85]	95.0	96.4	-	-
Zhao and Kit [87]	95.4	**97.6**	96.1	**95.7**
Sun et al. [75]	95.2	97.3	-	-
Sun et al. [76]	95.4	97.4	-	-
Zhang et al. [84]♣	96.1	97.4	-	-
Chen et al. [13]♠	94.5	95.4	-	-
Chen et al. [14]♠	94.8	95.6	-	-
Chen et al. [15]	94.3	96.0	-	94.8
Cai et al. [7]♢	95.8	97.1	95.6	95.3
baseline	95.2	97.3	95.1	94.9
+naive	90.5	92.1	91.3	94.3
+multi	**95.9**	97.4	**96.2**	95.4

Table 3.3: *Comparison with previous state-of-the-art models of results on all four Bakeoff-2005 datasets. Results with ♣ used external dictionary or corpus, with ♠ are from Cai and Zhao [6]'s runs on their released implementations without dictionary, with ♢ expurgated long words in test set.*

are not used when corpora are combined, in contrast to our method (+multi). The final F_1 scores are shown in Table 3.3.

According to this table, we find that naive way of mixing corpora brings much noise and errors, while our multi-criteria learning boosts performance on every single dataset. Compared to single-criterion learning models (baseline), multi-criteria learning model (+multi) outperforms all of them by up to 1.1%. Our joint model doesn't rob performance from one dataset to pay another, but share knowledge across datasets and improve performance on all datasets.

3.6.3 Results on SIGHAN bakeoff 2008

SIGHAN bakeoff 2008 [61] provided as many as 5 heterogeneous corpora. With another 3 non-repetitive corpora from SIGHAN bakeoff 2005, they form a large-scale standard dataset for multi-criteria CWS benchmark. We repeated our experiments

Models		MSR	AS	PKU	CTB	CKIP	CITYU	NCC	SXU	Avg.
Single-Criterion Learning										
Chen et al. [15]	P	95.70	93.64	93.67	95.19	92.44	94.00	91.86	95.11	93.95
	R	95.99	94.77	92.93	95.42	93.69	94.15	92.47	95.23	94.33
	F	95.84	94.20	93.30	95.30	93.06	94.07	92.17	95.17	94.14
Ours	P	97.17	95.28	94.78	95.14	94.55	94.86	93.43	95.75	95.12
	R	97.40	94.53	95.66	95.28	93.76	94.16	93.74	95.80	95.04
	F	97.29	94.90	95.22	95.21	94.15	94.51	93.58	95.78	95.08
Multi-Criteria Learning										
Chen et al. [15]	P	95.95	94.17	94.86	96.02	93.82	95.39	92.46	96.07	94.84
	R	96.14	95.11	93.78	96.33	94.70	95.70	93.19	96.01	95.12
	F	96.04	94.64	94.32	**96.18**	94.26	95.55	92.83	96.04	94.98
Ours	P	97.38	96.01	95.37	95.69	96.21	95.78	94.26	96.54	95.82
	R	97.32	94.94	96.19	96.00	95.27	95.43	94.42	96.44	95.64
	F	**97.35**	**95.47**	**95.78**	95.84	**95.73**	**95.60**	**94.34**	**96.49**	**95.73**

Table 3.4: *Results on test sets of 8 standard CWS datasets. Here, P, R, F indicate the precision, recall, F_1 value respectively. The maximum F_1 values are highlighted for each dataset.*

on these 8 corpora and compared our results with state-of-the-art scores, as listed in Table 3.4.

In the first block for single-criterion learning, we can see that our implementation is generally more effective than Chen et al. [15]'s. In the second block for multi-criteria learning, this disparity becomes even significant. And we further verified that every dataset benefit from our joint-learning solution. We also found that more data, even annotated with different standards or from different domains, brought better performance. Almost every dataset benefited from the larger scale of data. In comparison with large datasets, tiny datasets gained more performance growth.

3.7 Conclusions and Future Works

3.7.1 Conclusions

In this paper, we have presented a practical way to train multi-criteria CWS model. This simple and elegant solution only needs adding two artificial tokens at the beginning and ending of input sentence to specify the required target criterion. All the rest of model architectures, hyper-parameters, parameters and feature space are shared across all datasets. Experiments showed that our multi-criteria model can transfer knowledge between differently annotated corpora from heterogeneous domains. Our system is highly end-to-end, capable of learning large-scale datasets, and outperforms the latest state-of-the-art multi-criteria CWS works.

3.7.2 Future Works

Our effective and elegant multi-criteria learning solution can be applied to sequence labeling tasks such as POS tagging and NER. We plan to conduct more experiments of using our effective technique in various application domains.

CHAPTER 4

INDUSTRIAL STRENGTH DEPENDENCY PARSING SYSTEM

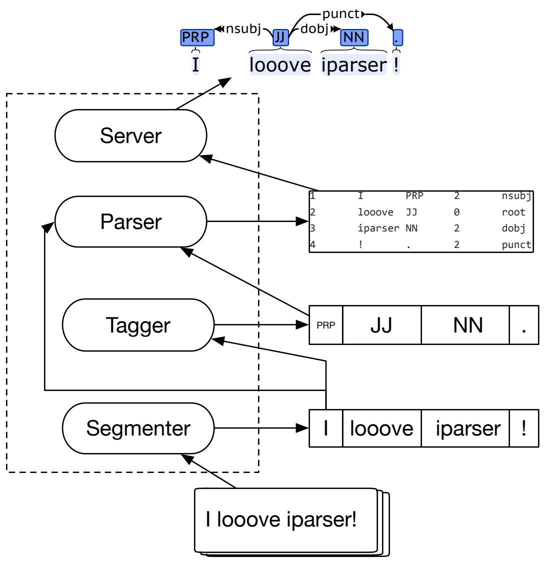
runtime speed We introduce the design and usage of IParser, an industrial strength multilingual parser for human language processing. This parser is capable of parsing raw text to dependency grammar trees, it also has tokenizers and part-of-speech taggers built-in. We address the runtime efficiency and memory usage over the accuracy digits after decimal mark. Our parser builds on recent state-of-the-art models, and comes with a friendly interface design. We evaluated several models and validated the minimal cost to build a concise network structure for production use. This implementation has been open sourced on GitHub[1].

4.1 Introduction

Dependency parsing is a useful and non-trial task among the many sub-tasks of Natural Language Processing (NLP). It is to analyze the dependency grammar structure of a given input sentence automatically. Many and various kinds of applications have exploited the power of dependency parsing, ranging from language modeling [11, 9], machine translation [20, 28, 3], question answering [19] to information retrieval [29]. For researchers and engineers, there has been some packages ready to use, including the well-known Stanford CoreNLP [55] for multilingual languages, the LTP [10] and HanLP [34] for Chinese language.

This paper describes yet another choice: IParser, a light-weighted Pythonic pipeline framework, which features with more recent models, easier installation, intergraded visualization tools and more friendly interfaces. IParser supports 3 common NLP

[1]`https://github.com/hankcs/iparser`



```
1       I        PRP     2       nsubj
2       looove   JJ      0       root
3       iparser  NN      2       dobj
4       !        .       2       punct
```

Figure 4.1: *Overall system architecture of IParser.*

steps: tokenization (word segmentation), part-of-speech tagging and dependency parsing, the main task.

We start with an overview of the system design and most distinguished features in section 2. Then we show simple usage and interface design in section 3. The algorithms, models and performance evaluations are given in section 4 and 5. We finish this paper with a brief conclusion and some tips for extending this parser.

4.2 System Design Overview

Our parser is a pipeline system, starting from raw text segmented to words, then tagged with part-of-speech labels, finally parsed to a dependency tree. The overview of architecture design is shown in Figure 4.4.

4.2.1 Pipeline Design

From bottom up, IParser transforms the input into temporary outputs, then the final dependency tree shown in user's browser. Our system is highly modularized. Every intermediate result can be retrieved without touching the blocks in higher level. This is important for users who only want partial results, like segmented words or tagged labels.

Although there is a trending to perform end-to-end dependency parsing jointly with segmentation and part-of-speech tagging [32], we didn't adopt such techniques. An end-to-end design can be expensive in corpora, and complex in system design.

Joint segmentation, tagging and parsing systems require all corpora to be fully annotated in dependency parsing level. But size of corpora tends to decrease with the increasing of task level, due to the increasing time and efforts for high level annotation. So, total amount of treebanks are relatively smaller than part-of-speech tagging corpora, and the later is generally smaller than segmentation corpora. Instead, pipeline systems can make use of rich corpora in lower level. Corpora in different levels don't have to come from the same source, thus they can be replaced flexibly.

In real life problems, joint learning systems suffer from the complexity both in implementation and runtime. They are highly end-to-end, require all tasks to be performed simultaneously. This requirement results in higher peak value of memory usage.

4.2.2 Multilingual Language

IParser is designed to be language-agnostic, it has universal language support. Users only need to feed it with some corpora of a desired language.

Although language-independent, we do provide mechanisms to perform preprocessing for particular languages, like the well-studied Chinese Word Segmentation. Besides, we provide pre-trained models for both English and Chinese, shipped in the installation package.

4.2.3 Interface Overview

Regarding user interface, we provide 3 types of interfaces:

1. Command Line Interface (CLI)
2. Application Interface (API)
3. Web based visualization and demonstration server (Server).

The API is designed for programmers who want to integrate IParser in their projects. It is flexible but requires some minimal Python codes. The CLI is intended for researchers for convenient access to segmentation, tagging and parsing functionalities. Then, the parsed dependency trees can be visualized by the server side, which generates a web page containing a tree structure in SVG (Scalable Vector Graphics) format. We will describe those interfaces in section 3.

4.3 Interfaces and Usage

The design goal of interfaces is to provide easy access to NLP pipelines with minimal effort, whether via API, CLI or browser. We hide most technique details in the configuration files, even provide default values for the only required parameter: the path to configuration file, which defaults to the bundled models.

4.3.1 Install

We have uploaded IParser distribution to PyPI[2] (the default Package Index for the Python community). So, it is possible to install via pip (the package management system for Python):

```
$ pip3 install iparser
```

Listing 4.1: *Install via pip*

The pre-trained models are bundled in Python package, ready for loading.

4.3.2 CLI

Getting started with one line of command is often attractive for new users. We provide such an on-the-fly command:

```
$ iparser segment <<< '商品和服务'
商品和服务

$ iparser tag <<< 'I looove iparser!'
I/PRP looove/VBP iparser/NN !/.

$ iparser parse <<< 'I looove iparser!'
1 I        _ _ PRP _ 2 nsubj _ _
2 looove   _ _ VBP _ 0 root  _ _
3 iparser  _ _ NN  _ 2 dobj  _ _
4 !        _ _ .   _ 2 punct _ _
```

Listing 4.2: *Get started*

[2]https://pypi.org/

Here user inputs some sentences, IParser performs segmentation, tagging and parsing for them. IParser is a compatible pipeline for standard I/O redirection. It is able to read inputs from console or files, and write outputs to them. By assembling these pipelines, user can handle normal NLP tasks without writing codes.

CLI can also help users with the training of new models. Every command (segment, tag, parse) share the same parameters for training. Take segmentation for an example, it only requires a configuration file from user:

```
$ iparser segment --help
usage: iparser segment [-h] [--config CONFIG] [--action ACTION]


optional arguments:
  -h, --help        show this help message and exit
  --config CONFIG   path to config file
  --action ACTION   which action (train, test, predict)?
```

Listing 4.3: *CLI for Training*

The required format of corpus varies with tasks. Generally speaking, segmentation and part-of-speech tagging are treated as tagging problem, so they share the same format. The format of dependency parsing corpora is CoNLL [5]. Regarding how to prepare corpora in those formats, readers are suggested to make use of the open source project TreebankPreprocessing[3].

[3]https://github.com/hankcs/TreebankPreprocessing

4.3.3 API

IParser provides full support for training, evaluation and prediction in Python API. We gathered everything in a single Python package called iparser. Minimal code for Python users are demonstrated in List 4.4:

```
$ python3
>>> from iparser import *
>>> iparser = IParser(pos_config_file=PTB_POS, dep_config_file=
   PTB_DEP)
>>> print(iparser.tag('I looove iparser!'))
[('I', 'PRP'), ('looove', 'VBP'), ('iparser', 'NN'), ('!', '.')]
>>> print(iparser.parse('I looove iparser!'))
1 I       _ _ PRP _ 2 nsubj _ _
2 looove  _ _ VBP _ 0 root  _ _
3 iparser _ _ NN  _ 2 dobj  _ _
4 !       _ _ .   _ 2 punct _ _
```

Listing 4.4: *Minimal Python Code*

Here PTB_POS is the path to configuration file of part-of-speech tagging model trained on Penn Treebank [56], and PTB_DEP stands for dependency parsing model.

User can load models trained on different corpora to support multilingual languages:

```
>>> iparser = IParser(seg_config_file=CTB_SEG, pos_config_file=
   CTB_POS, dep_config_file=CTB_DEP)
>>> print(iparser.parse('我爱依存分析！'))
1 我   _  _ PN  _ 2 nsubj _ _
2 爱   _  _ VV  _ 0 root  _ _
3 依存 _  _ VV  _ 2 ccomp _ _
```

```
4 分析 _   _ VV   _ 3 comod _ _
5 !     _   _ PU   _ 2 punct _ _
```

Listing 4.5: *Support for Chinese*

Those variables with names starting with `CTB_` are the configuration files to models trained on Chinese Treebank [81].

For users who don't need such an all-in-one interface, they can use indivisual interfaces for each task, as introduced in the next chapters.

4.3.3.1 Word Segmentation

The built-in word segmentation can be invoked via class `Segmenter`:

```
$ python3
>>> from iparser import *
>>> segmenter = Segmenter(CTB_SEG).load()
>>> segmenter.segment('下雨天地面积水')
['下雨天', '地面', '积水']
```

Listing 4.6: *Word Segmentation*

Notice that users should remember to call `load` in order to ensure a pre-trained model is loaded, otherwise an empty model will be created for training.

4.3.3.2 Part-of-Speech Tagging

The tagger `POSTagger` shares similar interfaces with `Segmenter`:

```
$ python3
>>> from iparser import *
>>> tagger = POSTagger(PTB_POS).load()
>>> tagger.tag('I looove languages'.split())
```

```
[('I', 'PRP'), ('looove', 'VBP'), ('languages', 'NNS')]
```

Listing 4.7: Part-of-Speech Tagging

POSTagger is not responsible for word segmentation. User should perform segmentation in advance or use IParser for convenience.

4.3.3.3 Dependency Parsing

The parser can be called through class DepParser:

```
$ python3
>>> from iparser import *
>>> parser = DepParser(PTB_DEP).load()
>>> sentence = [('Is', 'VBZ'), ('this', 'DT'), ('the', 'DT'), ('
   future', 'NN'), ('of', 'IN'), ('chamber', 'NN'), ('music', 'NN
   '), ('?', '.')]
>>> print(parser.parse(sentence))
1 Is      _ _ VBZ _ 4 cop _ _
2 this    _ _ DT  _ 4 nsubj _ _
3 the     _ _ DT  _ 4 det _ _
4 future  _ _ NN  _ 0 root  _ _
5 of      _ _ IN  _ 4 prep  _ _
6 chamber _ _ NN  _ 7 nn  _ _
7 music   _ _ NN  _ 5 pobj  _ _
8 ?       _ _ .   _ 4 punct _ _
```

Listing 4.8: Dependency Parsing

DepParser is neither responsible for segmentation nor tagging. The input should be a list of tuples, each tuple is consisted of word and tag.

4.3.3.4 Training and Evaluating

The training APIs for segmentation, part-of-speech tagging and dependency parsing are similar to each other. We take dependency parsing for an example. Assume that we have already prepared a configuration file `config_file` and corresponding corpus, the training can be done in one line of code:

```
parser = DepParser(config_file).train()
```

Listing 4.9: *Train a Dependency Parser*

We can also evaluate it on-the-fly:

```
parser.evaluate()
```

Listing 4.10: *Evaluate a Dependency Parser*

The outputs of `evaluate()` are the UAS (Unlabeled Attachment Score) and LAS (Labeled Attachment Score). Those scores will be stored in the log file `test.log` in the same folder with model.

4.3.4 Visualization

We have implemented and intergraded a http server for visualization of dependency trees. This server can be started via command line:

```
$ iparser serve --help
usage: iparser serve [-h] [--port PORT]

A http server for IParser

optional arguments:
  -h, --help    show this help message and exit
```

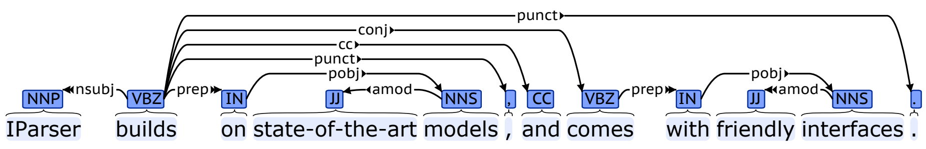


Figure 4.2: *A tree generated by IParser server.*

```
  --port PORT
```

Listing 4.11: *Visualization server*

We also provide a public server which is hosted at `http://iparser.hankcs.com/`. Users can input a sentence, hit enter to get the visualization of dependency tree in SVG format. A typical tree is illustrated in Figure 4.2.

4.3.5 Configuration File

Many parsers or softwares in academia often put all parameters in command line, requiring users to execute commands containing exactly the same parameters during the training and testing. This can lead to several issues. Firstly, one may need to type the parameters every time in terminal, bringing unnecessary efforts. Secondly, those parameters must be stored along with the serialized models, in order to keep model consistent when loading models, resulting extra efforts for maintenance of parameters. Finally, shell systems vary a lot, which may not ensure the universality of commands.

Instead, IParser employs configuration files to ensure the same network is created before and after serialization, in training phase and testing phase accordingly. This is important for research engineers who want to fine-tune those hyper parameters, or train new models on other corpora of a third language. We provide well documented configuration template files, containing all configurable parameters for users to adjust. In this way, the only parameter users need to specify in command line is the path to the configuration file. Along with serialization of model, the configuration file will be copied to the same place where model is stored. So, the meta info contained in configuration can easily be distributed with model itself.

A typical configuration file is shown in Listing 4.12.

```
[Data]
data_dir = data/ptb/pos
train_file = %(data_dir)s/train.tsv
dev_file = %(data_dir)s/dev.tsv
test_file = %(data_dir)s/test.tsv

[Save]
save_dir = iparser/static/ptb/pos
config_file = %(save_dir)s/config.ini
save_model_path = %(save_dir)s/model.txt
save_vocab_path = %(save_dir)s/vocab.pkl

[Network]
word_embedding_dim = 100
dropout = 0.2

[Optimizer]
learning_rate = 1e-2

[Run]
batch_size = 20
num_epochs = 60
```

Listing 4.12: *Configuration File*

4.3.6 Meta Information

The configuration of model is not the only meta information, we also distribute log files generated in training and testing along with serialized models. So, user can always check the statistics of training, development and test datasets, the performance of model on those datasets. A typical folder structure for a serialized model is shown in Listing 4.13.

```
iparser/static
|-- ptb
    |-- dep
        |-- config.ini
        |-- model.txt
        |-- test.log
        |-- train.log
        |-- vocab.pkl
```

Listing 4.13: *Folder Structure*

Here *config.ini* is the configuration file, *model.txt* and *vocab.pkl* are the serialized model and vocabulary respectively, *train.log* and *test.log* are the log files in training and testing phases.

	PTB		**CTB**	
Task	**Tagging**	**Parsing**	**Tagging**	**Parsing**
Train	0-18	02-21	001–270, 400–1151	001–815, 1001–1136
Dev	19-21	22	301–325	886–931, 1148–1151
Test	22-24	23	271-300	816–885, 1137–1147

Table 4.1: *Splits for part-of-speech tagging and dependency parsing. The split for tagging is taken from Collins [17], while the parsing splits are taken from Chen and Manning [12].*

4.4 Tagger for Segmentation and Part-of-Speech Tagging

In this chapter, we give a short review of the tagging algorithms.

4.4.1 Tagging Models

The tagging model is Bi-LSTM-CRF (bi-directional Long Short-term Memory Networks with Conditional Random Fields layer). Readers are suggested to refer to Huang et al. [40] and Lample et al. [51] for more information. We only give a short review for completeness.

LSTM Long Short-Term Memory Networks [38] are extensions of Recurrent Neural Networks (RNNs) designed to solve gradient vanishing issue by adding a memory-cell. This dedicated cell enables RNNs to capture long-range dependencies, which emancipates sequential model from fixed size windows.

Denote $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$ as a sequence of input vectors, the goal is to return another sequence $(\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_n)$ representing some information about the inputs at every time step. The following variant is adopted:

$$\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{W}_{xi}\mathbf{x}_t + \mathbf{W}_{hi}\mathbf{h}_{t-1} + \mathbf{W}_{ci}\mathbf{c}_{t-1} + \mathbf{b}_i) \\
\mathbf{c}_t &= (1 - \mathbf{i}_t) \odot \mathbf{c}_{t-1} + \\
&\qquad \mathbf{i}_t \odot \tanh(\mathbf{W}_{xc}\mathbf{x}_t + \mathbf{W}_{hc}\mathbf{h}_{t-1} + \mathbf{b}_c) \\
\mathbf{o}_t &= \sigma(\mathbf{W}_{xo}\mathbf{x}_t + \mathbf{W}_{ho}\mathbf{h}_{t-1} + \mathbf{W}_{co}\mathbf{c}_t + \mathbf{b}_o) \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t),
\end{aligned}$$

where σ is the element-wise sigmoid function.

Bi-LSTM Pair of forward and backward LSTM is called bidirectional LSTM (Bi-LSTM) [31] in literature. By concatenating the forward and backward hidden states, the final contextual representation is produced as $\mathbf{h}_t = [\overrightarrow{\mathbf{h}_t}; \overleftarrow{\mathbf{h}_t}]$.

CRF Conditional Random Fields are probabilistic graphical models which model the global tagging sequence instead of a local one. So adding a CRF layer can help tagging model to overcome label bias problem. Given a sequence $\mathbf{X}$ with n characters as $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n)$, CRF model tries to find the most possible tags $\mathbf{Y}^* = \{\mathbf{y}_1^*, \dots, \mathbf{y}_n^*\}$:

$$\mathbf{Y}^* = \mathrm{argmax}_{\mathbf{Y} \in \mathcal{T}^n} p(\mathbf{Y}|\mathbf{X}), \tag{4.1}$$

where $\mathcal{T}$ is the tagging set.

Character Model To solve the OOV (Out Of Vocabular) problem, rather than replace them with `UNK` symbol, we adopted the character LSTM layer introduced by Lample et al. [51] for part-of-speech tagging. This Bi-LSTM layer takes character embeddings as input, and concatenates the last hidden states of them to produce a vectorial representation for each word.

The part-of-speech tagging is already a tagging problem. Besides, we cast word segmentation into tagging problem following Xue [82]. The tag set for segmentation is $\mathcal{T} = \{\mathrm{B}, \mathrm{M}, \mathrm{E}, \mathrm{S}\}$, respectively standing for the **b**egin, **m**iddle, **e**nd of a word, or **s**ingle character forming a word.

4.4.2 Performance

We conducted experiments on PTB and CTB, using the conventional splits of dataset for each task.

CTB	Dev	Test
F_1	95.56	97.01

Table 4.2: *Segmentation score on CTB*

	PTB	CTB
Accuracy	97.39	95.69

Table 4.3: *Part-of-Speech tagging score on PTB and CTB*

4.4.2.1 Word Segmentation

For Chinese, we follow Jiang et al. [42], split the CTB into portions:

- Training: 001–270, 400–1151.
- Development: 301–325.
- Test: 271-300.

We adopted the radical enhanced character embeddings released by He et al. [36], which brings slightly better performance than embeddings trained via the conventional word2vec [60].

The performance of segmentation model is shown in Table 4.2.

For English, we used a rule-based tokenizer called segtok[4].

4.4.2.2 Part-of-Speech Tagging

The dataset splits are listed in Table 4.1.

For English, we exploited pre-trained GloVe embeddings [68]. For Chinese, we adopted embeddings trained by HanLP[5] on Chinese Wikipedia. The performances are recorded in Table 4.3.

[4]`https://github.com/fnl/segtok`
[5]`https://github.com/hankcs/HanLP/wiki/word2vec`

4.5 Dependency Parsing

	PTB		CTB	
Model	**UAS**	**LAS**	**UAS**	**LAS**
Bi-Affine	96.14	95.03	90.41	88.44
+char model	96.00	94.89	90.21	88.20

Table 4.4: *Test-set parsing results on English (PTB-SD 3.3.0) and Chinese (CTB-SD 3.3.0) datasets.*

In this section, we'll review the parsing models, then report performances on standard datasets.

4.5.1 Parsing Models

The implemented parsing model is the deep bi-affine model [23] with triple word embeddings [24].

4.5.1.1 Tripartie Word Embeddings

In NLP tasks, OOV often hurts performance on test set. Dozat et al. [24] designed 3 embeddings for each word.

Pre-trained Word Embedding This copy of pre-trained embedding $\mathbf{e}^{(pre)}$ won't be updated during the training, in order to retain similarity with its neighbor.

Randomly Initialized Word Embedding This randomly initialized embedding $\mathbf{e}^{(freq)}$ is trainable and shares the same dimension with pre-trained word embedding.

Character LSTM Hidden State with Attention For languages with rich morphology, a character level LSTM layer with attention mechanism is applied to each word, generating a hidden state for attention and the cell state. The concatenation of those two states is used as the final summary vector $\mathbf{v}^{(char)}$for each word.

The final word embedding $\mathbf{e}^{(word)}$ is computed as the element-wise sum of pre-trained embedding, the trainable embedding and character-level embedding:

$$\mathbf{e}^{(word)} = \mathbf{e}^{(pre)} + \mathbf{e}^{(freq)} + \mathbf{v}^{(char)} \tag{4.2}$$

This $\mathbf{e}_{word}$ will be concatenated together with POS embedding $\mathbf{e}_{tag}$.

$$\mathbf{x} = \mathbf{e}^{(word)} \oplus \mathbf{e}^{(tag)} \tag{4.3}$$

The resulting vector is used as the final input to the deep bi-affine parser in next section.

4.5.1.2 Deep Bi-Affine Parser

The first layer of this parser is a Bi-LSTM feature detector:

$$\mathbf{h}_i = \mathrm{BiLSTM}((\mathbf{x}_1, \cdots, \mathbf{x}_n))_i \tag{4.4}$$

where $(\mathbf{x}_1, \cdots, \mathbf{x}_n)$ is the vectorial representation of n words from a sentence.

We need to predict dependency arcs (arc) and relations (rel) given a pair of words as dependent or head, thus we have 4 kinds of predictions. So, 4 distinct multi-layer perceptrons (MLPs) are used to extract prediction related features from the raw feature $\mathbf{h}_i$:

$$\begin{aligned}
\mathbf{h}_i{}^{(arc-dep)} &= \mathrm{MLP}^{(arc-dep)}(\mathbf{h}_i) \\
\mathbf{h}_i{}^{(arc-head)} &= \mathrm{MLP}^{(arc-head)}(\mathbf{h}_i) \\
\mathbf{h}_i{}^{(rel-dep)} &= \mathrm{MLP}^{(rel-dep)}(\mathbf{h}_i) \\
\mathbf{h}_i{}^{(rel-head)} &= \mathrm{MLP}^{(rel-head)}(\mathbf{h}_i)
\end{aligned} \tag{4.5}$$

where $arc - dep$ means predicting the arc given the token i is dependent, and so on.

To make a prediction of what the arc is for token i, a bi-affine classifier is applied:

$$\begin{aligned}
\mathbf{s}_i{}^{(arc)} &= \begin{pmatrix} H^{(arc-head)} & \mathbf{1} \end{pmatrix} \\
&\quad \begin{pmatrix} W^{(arc)} \\ \mathbf{b}^{(arc)^T} \end{pmatrix} \mathbf{h}_i{}^{(arc-dep)} \\
y_i\prime^{(arc)} &= \arg\max_j s_{ij}{}^{(arc)}
\end{aligned} \tag{4.6}$$

where $H^{(arc-head)}$ is the stack of $\mathbf{h}_i{}^{(arc-dep)}$, $W^{(arc)}$ and $\mathbf{b}^{(arc)}$ are MLP parameters.

After the best arc $y_i\prime^{(arc)}$ is decided, a second bi-affine classifier is applied to predict the best label:

$$\begin{aligned}
\mathbf{s}_i{}^{(rel)} &= \mathbf{h}_{y_i\prime^{(arc)}}^{\top(rel-head)} \mathbf{U}^{(rel)} \mathbf{h}_i^{(rel-dep)} \\
&+ \begin{pmatrix} W^{(rel)} \\ \mathbf{b}^{(rel)^T} \end{pmatrix} \begin{pmatrix} \mathbf{h}_i{}^{(rel-dep)} & \\ \mathbf{h}_{y_i\prime^{(arc)}}^{(rel-head)} & \mathbf{1} \end{pmatrix} \\
y_i\prime^{(rel)} &= \arg\max_j s_{ij}{}^{(rel)}
\end{aligned} \tag{4.7}$$

where $\mathbf{U}^{(rel)} \in \mathbb{R}^{m \times k \times k}$ is a rank 3 tensor, m is the number of possible relations, k is the dimension of $\mathbf{h}_i{}^{(rel)}$.

The loss function is the sum of softmax cross-entropy losses of these two bi-affine classifiers:

$$CE(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_i y_i \times log(\hat{y_i}) \tag{4.8}$$

$$\begin{aligned} J = CE(\boldsymbol{y}^{(arc)}, \hat{\boldsymbol{y}}^{(arc)}) + \\ CE(\boldsymbol{y}^{(rel)}, \hat{\boldsymbol{y}}^{(rel)}) \end{aligned} \tag{4.9}$$

where $\mathbf{y}$ and $\boldsymbol{y}$ are predictions, $\hat{\mathbf{y}}$ and $\boldsymbol{y}$ are gold answers. For relation prediction, the gold relation is still used even if arc prediction is wrong.

4.5.2 Performance

On standard splits shown in Table 4.1, this implementation achieved scores listed in Table 4.4.

These scores are higher than reported in Dozat and Manning [23]'s original paper, we assume that the preprocessing is different. They omitted punctuations, while we kept them. Because IParser is intended for practical scenario, where punctuations are very common in natural languages.

The character model doesn't bring benefit for English and Chinese, but slows down runtime speed. So, we disabled it in the release version.

4.6 Conclusion

In this paper, we introduced the design, interfaces, algorithms and performances of IParser, an industrial strength dependency parsing pipeline. Rather than simply showing the codes, we have explained the motivations and reasons for the design, the underlying algorithms to support those features.

The last piece of puzzle is about how to extend IParser. Users are welcomed to upload models trained on more languages, and join the development on GitHub. We made the source available under the GNU General Public License v3.0 [6], with the purpose of encouraging scholars to release their codes along with papers.

Acknowledgments

During the development of IParser, we made use of some excellent open source projects. The whole implementation is built on top of DyNet [62], a deep learning framework featured with dynamic computation graph. We refactored Robert's Bi-LSTM-CRF implementation[7] to satisfy our design. We extended Deng Cai's parser implementation[8] to support character level model. For the visualization part, we adopted Annodoc[9], a annotation documentation support system which is used as a wrapper of BRAT [73]. IParser is simply an intergraded system of those cutting-edge tagger, parser and visualization tool.

[6]`https://github.com/hankcs/iparser/blob/master/LICENSE`
[7]`https://github.com/rguthrie3/BiLSTM-CRF`
[8]`https://github.com/jcyk/Dynet-Biaffine-dependency-parser`
[9]`https://github.com/spyysalo/annodoc`

CHAPTER 5
CONCLUSIONS AND FUTURE WORK

In this chapter, we summarize our contributions presented in this dissertation. We then discuss the possible directions for our future research work.

5.1 Summary

Dependency parsing is a non-trivial and useful task for Natural Language Processing. To build an industrial strength parsing system, we explored word segmentation and part-of-speech tagging in advance.

To make this study concrete, we implemented a pipeline parsing system called IParser, which is a lightweight and integrated parser with built-in tokenizer, tagger and visualization tool. We then open sourced IParser for academic or commercial use, and hosted a live demo website.

5.2 Future Work

Aiming at dependency parsing, IParser is designed as a pipeline system with replaceable sub-systems.

In the future, we can extend our research on a variety of models and algorithms, such as transition-based parsing, end-to-end or joint learning.

In summary, our research is of practical significance and has immediate relevance and huge impact in industry. We are looking forward to continuing and making an impact in the emerging NLP area.

BIBLIOGRAPHY

[1] Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. Globally normalized transition-based neural networks. *arXiv preprint arXiv:1603.06042*, 2016.

[2] Miguel Ballesteros, Yoav Goldberg, Chris Dyer, and Noah A Smith. Training with exploration improves a greedy stack-lstm parser. *arXiv preprint arXiv:1603.03793*, 2016.

[3] Antonio Valerio Miceli Barone and Giuseppe Attardi. Pre-reordering for machine translation using transition-based walks on dependency parse trees. In *Proceedings of the Eighth Workshop on Statistical Machine Translation*, pages 164–169, 2013.

[4] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching Word Vectors with Subword Information. *arXiv.org*, page arXiv:1607.04606, July 2016.

[5] Sabine Buchholz and Erwin Marsi. Conll-x shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, pages 149–164. Association for Computational Linguistics, 2006.

[6] Deng Cai and Hai Zhao. Neural Word Segmentation Learning for Chinese. *ACL*, 2016.

[7] Deng Cai, Hai Zhao, Zhisong Zhang, Yuan Xin, Yongjian Wu, and Feiyue Huang. Fast and Accurate Neural Word Segmentation for Chinese. *arXiv.org*, page arXiv:1704.07047, April 2017.

[8] Jiayuan Chao, Zhenghua Li, Wenliang Chen, and Min Zhang. Exploiting Heterogeneous Annotations for Weibo Word Segmentation and POS Tagging. *NLPCC*, 2015.

[9] Eugene Charniak et al. Parsing as language modeling. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2331–2336, 2016.

[10] Wanxiang Che, Zhenghua Li, and Ting Liu. LTP - A Chinese Language Technology Platform. *COLING*, 2010.

[11] Ciprian Chelba, David Engle, Harry Printz, Frederick Jelinek, Eric Ristad, Victor Jimenez, Ronald Rosenfeld, Sanjeev Khudanpur, Andreas Stolcke, Lidia Mangue, et al. Structure and performance of a dependency language model. Technical report, SRI INTERNATIONAL MENLO PARK CA SPEECH TECHNOLOGY AND RESEARCH LAB, 1997.

[12] Danqi Chen and Christopher Manning. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 740–750, 2014.

[13] Xinchi Chen, Xipeng Qiu, Chenxi Zhu, and Xuanjing Huang. Gated Recursive Neural Network for Chinese Word Segmentation. *ACL*, 2015.

[14] Xinchi Chen, Xipeng Qiu, Chenxi Zhu, Pengfei Liu, and Xuanjing Huang. Long Short-Term Memory Neural Networks for Chinese Word Segmentation. *EMNLP*, 2015.

[15] Xinchi Chen, Zhan Shi, Xipeng Qiu, and Xuanjing Huang. Adversarial Multi-Criteria Learning for Chinese Word Segmentation. 1704:arXiv:1704.07556, 2017.

[16] Hao Cheng, Hao Fang, Xiaodong He, Jianfeng Gao, and Li Deng. Bi-directional attention with agreement for dependency parsing. *arXiv preprint arXiv:1608.02076*, 2016.

[17] Michael Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 1–8. Association for Computational Linguistics, 2002.

[18] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel P Kuksa. Natural Language Processing (Almost) from Scratch. *Journal of Machine Learning Research*, 2011.

[19] Hang Cui, Renxu Sun, Keya Li, Min-Yen Kan, and Tat-Seng Chua. Question answering passage retrieval using dependency relations. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 400–407. ACM, 2005.

[20] Yuan Ding and Martha Palmer. Synchronous dependency insertion grammars: A grammar formalism for syntax based statistical mt. In *Proceedings of the Workshop on Recent Advances in Dependency Grammar*, 2004.

[21] Chuanhai Dong, Jiajun Zhang, Chengqing Zong, Masanori Hattori, and Hui Di. Character-Based LSTM-CRF with Radical-Level Features for Chinese Named Entity Recognition. *NLPCC/ICCPOL*, 2016.

[22] Timothy Dozat and Christopher D Manning. Deep Biaffine Attention for Neural Dependency Parsing. *CoRR*, cs.CL:arXiv:1611.01734, 2016.

[23] Timothy Dozat and Christopher D Manning. Deep biaffine attention for neural dependency parsing. *arXiv preprint arXiv:1611.01734*, 2016.

[24] Timothy Dozat, Peng Qi, and Christopher D Manning. Stanford's graph-based neural dependency parser at the conll 2017 shared task. *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 20–30, 2017.

[25] Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A Smith. Transition-based dependency parsing with stack long short-term memory. *arXiv preprint arXiv:1505.08075*, 2015.

[26] Jason M Eisner. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th conference on Computational linguistics-Volume 1*, pages 340–345. Association for Computational Linguistics, 1996.

[27] Thomas Emerson. The second international chinese word segmentation bakeoff. In *Proceedings of the Fourth SIGHAN Workshop on Chinese Language Processing*, pages 123–133. Jeju Island, Korea, 2005.

[28] Michel Galley and Christopher D Manning. Quadratic-time dependency parsing for machine translation. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, pages 773–781. Association for Computational Linguistics, 2009.

[29] Pablo Gamallo, Marcos Garcia, and Santiago Fernández-Lanza. Dependency-based open information extraction. In *Proceedings of the Joint Workshop on Unsupervised and Semi-Supervised Learning in NLP*, pages 10–18. Association for Computational Linguistics, 2012.

[30] Yoav Goldberg and Joakim Nivre. A dynamic oracle for arc-eager dependency parsing. In *COLING*, pages 959–976, 2012.

[31] Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, 18(5-6):602–610, July 2005.

[32] Kazuma Hashimoto, Caiming Xiong, Yoshimasa Tsuruoka, and Richard Socher. A joint many-task model: Growing a neural network for multiple nlp tasks. *arXiv preprint arXiv:1611.01587*, 2016.

[33] Aboul Ella Hassanien, Khaled Shaalan, Tarek Gaber, Ahmad Taher Azar, and Fahmy Tolba. *Proceedings of the International Conference on Advanced Intelligent Systems and Informatics 2016*, volume 533. Springer, 2016.

[34] Han He. HanLP: Han Language Processing, 2014. URL `https://github.com/hankcs/HanLP`.

[35] Han He, Lei Wu, Xiaokun Yang, Hua Yan, Zhimin Gao, Yi Feng, and George Townsend. Analytical study of dependency parsing. *3rd International Conference on Intelligent Computing, Communication & Devices (ICCD-2017)*, 2017.

[36] Han He, Lei Wu, Xiaokun Yang, Hua Yan, Zhimin Gao, Yi Feng, and George Townsend. Dual long short-term memory networks for sub-character representation learning. *arXiv preprint arXiv:1712.08841*, 2017.

[37] Han He, Lei Wu, Hua Yan, Zhimin Gao, Yi Feng, and George Townsend. Effective neural solution for multi-criteria word segmentation. *2nd International Conference on Smart Computing & Informatics (SCI-2018)*, 2018.

[38] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[39] C. Huang and H. Zhao. Chinese word segmentation: A decade review. *Journal of Chinese Information Processing*, 21(3):8–19, 2007.

[40] Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*, 2015.

[41] Wenbin Jiang, Liang Huang, and Qun Liu. Automatic adaptation of annotation standards: Chinese word segmentation and POS tagging. In *the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference*, pages 522–530, Morristown, NJ, USA, 2009. Association for Computational Linguistics.

[42] Wenbin Jiang, Liang Huang, and Qun Liu. Automatic adaptation of annotation standards: Chinese word segmentation and pos tagging: a case study. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1*, pages 522–530. Association for Computational Linguistics, 2009.

[43] Kiat Low Jin, Hwee Tou Ng, and Wenyuan Guo. A maximum entropy approach to chinese word segmentation. *Proceedings of the Fourth Sighan Workshop on Chinese Language Processing*, 2005.

[44] Melvin Johnson, Mike Schuster, Quoc V Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda B Viégas, Martin Wattenberg, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's Multilingual Neural Machine Translation System - Enabling Zero-Shot Translation. cs.CL, 2016.

[45] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. Character-aware neural language models. In *AAAI*, pages 2741–2749, 2016.

[46] Eliyahu Kiperwasser and Yoav Goldberg. Simple and Accurate Dependency Parsing Using Bidirectional LSTM Feature Representations. *arXiv.org*, page arXiv:1603.04351, March 2016.

[47] Sandra Kübler, Ryan McDonald, and Joakim Nivre. Dependency parsing. *Synthesis Lectures on Human Language Technologies*, 1(1):1–127, 2009.

[48] Marco Kuhlmann, Carlos Gómez-Rodríguez, and Giorgio Satta. Dynamic programming algorithms for transition-based dependency parsers. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 673–682. Association for Computational Linguistics, 2011.

[49] Adhiguna Kuncoro, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, Graham Neubig, and Noah A Smith. What do recurrent neural network grammars learn about syntax? *arXiv preprint arXiv:1611.05774*, 2016.

[50] John D. Lafferty, Andrew Mccallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Eighteenth International Conference on Machine Learning*, pages 282–289, 2001.

[51] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural Architectures for Named Entity Recognition. *CoRR*, 2016.

[52] Phong Le and Willem Zuidema. The inside-outside recursive neural network model for dependency parsing. In *Emnlp*, pages 729–739, 2014.

[53] Yanran Li, Wenjie Li, Fei Sun, and Sujian Li. Component-Enhanced Chinese Character Embeddings. *EMNLP*, 2015.

[54] Zhenghua Li, Jiayuan Chao, Min Zhang, and Wenliang Chen. Coupled Sequence Labeling on Heterogeneous Annotations: POS Tagging as a Case Study. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1783–1792, Stroudsburg, PA, USA, 2015. Association for Computational Linguistics.

[55] Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. The Stanford CoreNLP Natural Language Processing Toolkit. *ACL*, 2014.

[56] Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.

[57] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, page 3, 2010.

[58] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. *arXiv.org*, January 2013.

[59] Tomas Mikolov, Ilya Sutskever, Kai Chen 0010, Gregory S Corrado, and Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality. *NIPS*, 2013.

[60] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[61] PRC MOE. The fourth international chinese language processing bakeoff: Chinese word segmentation, named entity recognition and chinese pos tagging. In *Proceedings of the sixth SIGHAN workshop on Chinese language processing*, 2008.

[62] Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, et al. Dynet: The dynamic neural network toolkit. *arXiv preprint arXiv:1701.03980*, 2017.

[63] Joakim Nivre. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT*. Citeseer, 2003.

[64] Joakim Nivre. Incrementality in deterministic dependency parsing. In *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*, pages 50–57. Association for Computational Linguistics, 2004.

[65] Wenzhe Pei, Tao Ge, and Baobao Chang. Max-Margin Tensor Neural Network for Chinese Word Segmentation. *ACL*, 2014.

[66] Wenzhe Pei, Tao Ge, and Baobao Chang. An effective neural network model for graph-based dependency parsing. In *ACL (1)*, pages 313–322, 2015.

[67] Fuchun Peng, Fangfang Feng, and Andrew Mccallum. Chinese segmentation and new word detection using conditional random fields. pages 562–568, 2004.

[68] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[69] Yuval Pinter, Robert Guthrie, and Jacob Eisenstein. Mimicking word embeddings using subword rnns. *arXiv preprint arXiv:1707.06961*, 2017.

[70] Yanjun Qi, Sujatha G Das, Ronan Collobert, and Jason Weston. Deep Learning for Character-Based Information Extraction. *ECIR*, 2014.

[71] Xipeng Qiu, Jiayi Zhao, and Xuanjing Huang. Joint Chinese Word Segmentation and POS Tagging on Heterogeneous Annotated Corpora with Multiple Task Learning. *EMNLP*, 2013.

[72] Xinlei Shi, Junjie Zhai, Xudong Yang, Zehua Xie, and Chao Liu. Radical Embedding - Delving Deeper to Chinese Radicals. *ACL*, 2015.

[73] Pontus Stenetorp, Sampo Pyysalo, Goran Topić, Tomoko Ohta, Sophia Ananiadou, and Jun'ichi Tsujii. Brat: a web-based tool for nlp-assisted text annotation. In *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 102–107. Association for Computational Linguistics, 2012.

[74] Weiwei Sun and Xiaojun Wan. Reducing Approximation and Estimation Errors for Chinese Lexical Processing with Heterogeneous Annotations. *ACL*, 2012.

[75] Xu Sun, Yaozhong Zhang, Takuya Matsuzaki, Yoshimasa Tsuruoka, and Junichi Tsujii. A discriminative latent variable chinese segmenter with hybrid word/character information. pages 56–64, 2009.

[76] Xu Sun, Houfeng Wang, and Wenjie Li. Fast online training with frequency-adaptive learning rates for chinese word segmentation and new word detection. pages 253–262, 2012.

[77] Yaming Sun, Lei Lin, Nan Yang, Zhenzhou Ji, and Xiaolong Wang. Radical-Enhanced Chinese Character Embedding. *ICONIP*, 8835(Chapter 34):279–286, 2014.

[78] Ben Taskar, Vassil Chatalbashev, Daphne Koller, and Carlos Guestrin. Learning structured prediction models: A large margin approach. In *Proceedings of the 22nd international conference on Machine learning*, pages 896–903. ACM, 2005.

[79] Huihsin Tseng, Pichuan Chang, Galen Andrew, Daniel Jurafsky, and Christopher Manning. A conditional random field word segmenter for sighan bakeoff 2005. pages 168–171, 2005.

[80] David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. Structured training for neural network transition-based parsing. *arXiv preprint arXiv:1506.06158*, 2015.

[81] Naiwen Xue, Fei Xia, Fu-Dong Chiou, and Marta Palmer. The penn chinese treebank: Phrase structure annotation of a large corpus. *Natural language engineering*, 11(2):207–238, 2005.

[82] Nianwen Xue. Chinese Word Segmentation as Character Tagging. *IJCLCLP*, 2003.

[83] Hiroyasu Yamada and Yuji Matsumoto. Statistical dependency analysis with support vector machines. In *Proceedings of IWPT*, volume 3, pages 195–206, 2003.

[84] Longkai Zhang, Houfeng Wang, Xu Sun, and Mairgup Mansur. Exploring representations from unlabeled data with co-training for chinese word segmentation. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, page 311 – 321, Seattle, Washington, USA, October 2013. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/D13-1031`.

[85] Yue Zhang and Stephen Clark. Chinese segmentation with a word-based perceptron algorithm. pages 840–847, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P/P07/P07-1106`.

[86] Yue Zhang and Joakim Nivre. Transition-based dependency parsing with rich non-local features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 188–193. Association for Computational Linguistics, 2011.

[87] Hai Zhao and Chunyu Kit. Unsupervised segmentation helps supervised learning of character tagging for word segmentation and named entity recognition. In *The Sixth SIGHAN Workshop on Chinese Language Processing*, page 106–111, 2008.

[88] Hai Zhao, Changning Huang, Mu Li, and Bao-Liang Lu. Effective Tag Set Selection in Chinese Word Segmentation via Conditional Random Field Modeling. *PACLIC*, 2006.

[89] Hai Zhao, Chang Ning Huang, Mu Li, and Bao Liang Lu. A unified character-based tagging framework for chinese word segmentation. *Acm Transactions on Asian Language Information Processing*, 9(2):1–32, 2010.

[90] Xiaoqing Zheng, Hanyang Chen, and Tianyu Xu. Deep Learning for Chinese Word Segmentation and POS Tagging. *EMNLP*, 2013.

VITA

Han He

2011	BEc, Japanese Culture and Economics Shanghai International Studies University Shanghai, China

FIRST AUTHORED PUBLICATIONS

1. H. He, L. Wu, H. Yan, Z. Gao, Y. Feng, G. Townsend, "Effective Neural Solution for Multi-Criteria Word Segmentation", 2nd International Conference on Smart Computing & Informatics (SCI-2018), Springer Smart Innovation Systems and Technologies Book Series, Springer-Verlag, 2018
2. H. He, L. Wu, H. Yan, Y. Feng, "Analytical Study of Dependency Parsing ", 3rd International Conference on Intelligent Computing, Communication & Devices (ICCD-2017), Springer Advances in Intelligent Systems and Computing Book Series, Springer-Verlag, 2017
3. H. He, L. Wu, X. Yang, H. Yan, Z. Gao, Y. Feng, G. Townsend, "Dual Long Short-Term Memory Networks for Sub-Character Representation Learning", The 15th International Conference on Information Technology - New Generations (ITNG-2018); Book Chapter Paper, Springer Book Series: Advances in Intelligent Systems & Computing, Springer-Verlag, April 2018