NAIST-IS-DD1561004

Doctoral Dissertation

A Study on Syntactic and Semantic Dependency Parsing

Hiroki Ouchi

February 1, 2018

Department of Information Processing
Graduate School of Information Science
Nara Institute of Science and Technology

A Doctoral Dissertation
submitted to Graduate School of Information Science,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
Doctor of ENGINEERING

Hiroki Ouchi

Thesis Committee:

Professor Yuji Matsumoto	(Supervisor)
Professor Satoshi Nakamura	(Co-supervisor)
Associate Professor Masashi Shimbo	(Co-supervisor)
Assistant Professor Hiroyuki Shindo	(Co-supervisor)
Assistant Professor Hiroshi Noji	(Co-supervisor)

A Study on Syntactic and Semantic Dependency Parsing*

Hiroki Ouchi

Abstract

Syntactic and semantic dependency parsing are a fundamental problem in natural language processing (NLP). A variety of techniques have been proposed for improving syntactic and semantic dependency parsers. However, there still remains some room for further improvement. This thesis describes several methods for improving syntactic and semantic dependency parsing.

To improve syntactic dependency parsing, we design and use *supertags*. Supertags, which are lexical templates extracted from dependency structure annotated corpus, encode linguistically rich information that imposes complex constraints in a local context. We present a supertag design framework that allows us to design various granularity-level supertag sets. To investigate the appropriate granularity or design of supertags needed to improve parsing performance, we build various supertag sets based on the framework. Then, using the supertag sets as features, we perform experiments on multilingual syntactic dependency parsing. The experimental results show that appropriately designed supertags are effective for syntactic dependency parsing.

To improve semantic dependency parsing, we capture and exploit *multi-predicate interactions*. This approach is based on the linguistic intuition that the predicates in a sentence are semantically related to each other, and capturing this relation can be useful for semantic dependency parsing. To capture this information, we propose two distinct methods using (i) bipartite graphs and (ii) grid-type recurrent neural networks. Performing experiments on Japanese predicate argument

*Doctoral Dissertation, Department of Information Processing, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DD1561004, February 1, 2018.

structure analysis, we demonstrate that our proposed methods yield considerable improvements.

Keywords:

Syntactic Dependency Parsing, Semantic Dependency Parsing, Predicate Argument Structure Analysis, Supertags, Bipartite Graphs, Recurrent Neural Networks

Acknowledgments

主指導教員の松本裕治教授には，この研究室に私を受け入れていただきました．いつ頃からか，ゼミ室で行われる勉強会や研究会のとき，私は松本先生の隣の席に座るようになりました．間近で先生のお話を聞かせていただいたり，他愛もない雑談をさせていただき，いつも心地よさを感じています．5 年の間に，松本研 OB の方や松本先生と関わりのあるたくさんの方と繋がり，楽しい時間を過ごしました．卒業が近づき，奈良で過ごしたひとかたまりの時間が無数の断片のようなものになりつつあります．これから先，ふとした時に，一つ一つの断片を手にとって，確かな重みを感じながら振り返ることを楽しみにしています．

中村哲教授には，お忙しい中，副指導教員を引き受けていただきました．中村先生の助言により，論文の完成度が増したと思います．私が中村先生に最も接近したのは，2015 年に ACL が北京で開催されたときです．その日のお昼ご飯にご一緒させていただいたことを記憶しております．これからの研究の話や中村先生のスマートな立ち振る舞いに感銘を受けました．

新保仁准教授からはユーモアのセンスを学びました．授業や研究発表の冒頭，新保先生が何気なく添える話を，私はすごく面白いと思っていました．たまに私も研究発表の冒頭にアイスブレイクの話を入れますが，その構成や雰囲気は新保先生に大きく影響を受けています．新保先生との直接的な関わりは，DMLA が主だったと思います．矢のようなご指摘は，私の研究の曖昧な点を正確に射抜きました．また，私に対してのみならず他の学生にも，各人の内面的適正を考慮しながらご指導されていた姿が強く印象に残っています．研究者としてだけでなく指導者としての新保先生にも大きな尊敬の念を抱いております．

進藤裕之助教には研究を行う活力をいただきました．深夜，助教室から灯りが漏れてきて，それに群がる蛾のように，私はミーティングをさせていただくためにドアを叩きます．ときにその灯りは陽光のようでもあり，私にエネルギーを与え，もう一度机に向かわせます．数多くの助言もさることながら，無心に研究を

する進藤先生の姿に，私は不思議な安心感を抱いていました．今後も，進藤先生のスタイルを一つの見本として邁進して行きたいと思います．

能地宏助教には，2014 年の言語処理学会年次大会で初めてお会いしました．そのときに，「Supertag の論文，よかったよ。」と話しかけていただいたことを覚えております．自分が書いた論文が誰かに読まれていることを初めて実感した瞬間でした．嬉しさと微妙な照れくささが混ざり合ったような心境でした．お酒をご一緒させていただく機会では，構文解析に関する幅広いご意見を聞かせていただきました．時折，明日の予定を顧みずにお酒を飲み進める能地さんの姿に勇気づけられることもありました．これからもよろしくお願いします．

Johns Hopkins University の Kevin Duh 先生には，私が右も左もわからない状態のときに，研究のいろはを教えていただきました．私が M1 のときに初めて国際会議に論文を投稿した際も，Kevin 先生の助けがなければ，論文を書き上げられなかったと思います．アメリカの大学に移られてからも，国際会議でお会いするたびに気にかけていただき，心強く感じます．会議中に各国の研究者と楽しそうに議論をされている Kevin 先生の姿に，世界を舞台にして研究を行う醍醐味を感じました．

北川祐子秘書には事務手続きでお世話になりました．提出期限を過ぎた書類にもおおらかに対応するその寛容さは，提出が遅れてもなんとかなるんだと，私に勇気を与えました．また，他愛もない世間話から人生に関わる壮大な雑談まで，多様な話題を共有し，いくつもの助言をいただきました．北川さんに研究室の環境を整えていただき，楽しく研究活動が行えた 5 年間でした．

Preferred Networks の坪井祐太さんには，2015 年の夏から共同研究をさせていただきました．2015 年春，当時坪井さんが在籍されて居た IBM 東京基礎研究所のインターンシップに応募したことは，私が博士課程において行った最高の選択のひとつです．夏のインターンシップでメンターを引き受けていただき，毎日のように研究の相談をさせていただきました．研究を進めていく過程自体に大きな楽しみがあり，良い研究を進めていく上で必要な定石のようなものを学ばせていただいたと思います．

知能コミュニケーション研究室の吉野幸一郎助教には，研究に関する数多くの助言をいただきました．特に，私が D3 になってからは共同研究をさせていただ

き，定期的にミーティングをさせていただきました．研究に関する話だけでなく，日本の自然言語処理の現状や野球の話など，様々な話題を共有していただき，とても和みました．

情報通信研究機構の藤田篤主任研究員と飯田龍主任研究員には，研究内容だけでなく，社会に出てからの研究との向き合い方など，多岐にわたる助言をいただきました．これからの自分の研究生活を考えるきっかけになりました．

東京 D の会のメンバーには多くの刺激をいただきました．異なる研究室の同じ博士課程の学生として共有している部分も多く，大きな励みになりました．

松本研のメンバーにはとてもお世話になりました．私は自分の母親に会うとき，よく松本研のメンバーの話をします．

登場回数第一位は澤井裕一郎くんです．「北京空港着いたら澤井くんがタクシー運転手に中国語で話しかけて…」「なんか澤井くんが角田市のこと知ってたよ。」いろんなエピソードを話しました．澤井くんが自分語りをすることはほとんどないけれど，掘れば掘るほど何かが湧き出てくる，豊かな水脈を持っているんだなあと私は思っています．機会があったら，私の母親に一度会ってあげてください．

先輩の濱口拓男さんには萌芽的な研究の話をたくさん聞かせていただきました．ニューラルネットを全身に纏った濱口さんの壮大な理論は，ときに困惑を，ときに感動を私に与えました．一方，今忙しくないですか，今聞く気分じゃないですか，といったことを必ず確認してくれるところに，細かな気遣いも感じました．大胆さと繊細さを兼ね備えた濱口さんの今後の研究が楽しみです．

OB の重藤優太郎さんと椿真史さんにもたくさんのご指導を受けました．特に，お二人の DMLA でのご活躍が強く印象に残っております．お二人が卒業する際，来年はどうなるのだろうと一抹の不安を覚えるくらい，私にとって大きな存在でした．

他の OB の先輩にもお世話になりました．特に，5 階に住んでいた先輩方には感銘を受けました．本当にこのような人物が実在するのかと目を疑いました．そこにはアニメの世界がオーバーレイしたような空間が形成され，部屋の向こう側とこちら側では空間の解像度が違って見えました．

卒業した後輩の皆さまにも多くの刺激を受けました．特に，三田雅人くんと駒井雅之くんにはたびたび居酒屋に付き合ってもらいました．日を追うごとに成長していくお二人を頼もしく思いました．二人が松本研で研究している姿をもう少し長く見ていたかったです．あっという間に二年は過ぎ去りました．他にも，少し変わった後輩が毎年のように入学してきて，松本研を彩っていきました．

同期入学のみなさまとは，NAIST 入学以来，多くの時間を共有しました．緩やかな関係性の中でのコミュニケーションに自然と肩の力が抜けました．特に，毎週金曜に催される飲み会は，深い安らぎをもたらしました．卒業してからも，時たま開かれる飲み会を楽しみにしています．

宮城にいる父，母，弟へ．奈良に流れ着き，早 5 年が経ちました．次はどこに流れていくかわかりませんが，これからも私を見守っていてください．

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Syntactic and semantic analysis are a fundamental problem in natural language processing (NLP). Dependency-based methods for the analysis have attracted considerable attention. The popularity stems from their easily interpretable encoding of syntactic and semantic structures.

1.1 Dependency Representations and Parsing

Syntactic Dependencies

Syntactic Dependency Representations

A dependency representation consists of words (or lexical elements) linked by binary asymmetric relations called *dependencies* [44]. Especially, in this thesis, dependencies which represent syntactic structure are called *syntactic dependencies*. A syntactic dependency holds between two words: one is called the *dependent* and another is called the *head*.

- **Dependent**: a syntactically subordinate word
- **Head**: the word on which a subordinate word depends

Consider the example sentence of Figure 1.1: "She makes and repairs computers." Each arc (drawn in green) above the sentence denotes a syntactic dependency. For example, the arc from "makes" to "She" with `subj` represents that a

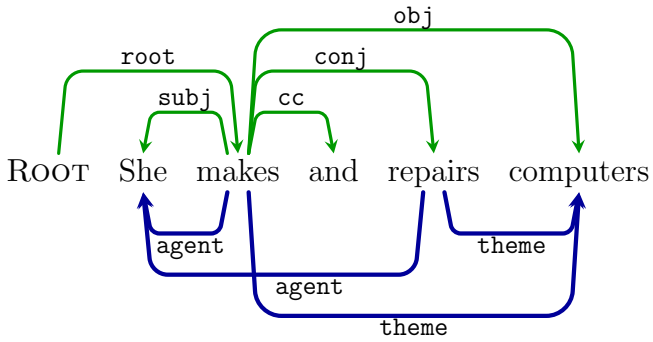


Figure 1.1: Example of syntactic and semantic dependencies, i.e., syntactic dependencies in the above part and semantic dependencies in the below part.

syntactically subordinate word "She" depends on another word "makes" with the dependency (grammatical) type "subject" (`subj`). All the syntactic dependencies are listed as follows:

$$\langle \text{makes}, \texttt{subj}, \text{She} \rangle, \langle \textsc{Root}, \texttt{root}, \text{makes} \rangle, \langle \text{makes}, \texttt{cc}, \text{and} \rangle,$$
$$\langle \text{makes}, \texttt{conj}, \text{repairs} \rangle, \langle \text{makes}, \texttt{obj}, \text{computers} \rangle$$

where each triple consists of ⟨head, `dependency type`, dependent⟩. Note that the head word ROOT of the word "makes" is a special word, and each word has a single head word. Such bilexical relations have been used to improve performance of NLP applications such as machine translation and information extraction.

Syntactic Dependency Parsing

A task of recovering the syntactic structure of a sentence is called *syntactic dependency parsing*. Most of recent methods for the task use *machine learning* techniques. In particular, *supervised* methods have attracted the most attention. They presuppose that there is a training set:

$$\mathcal{D}^{train} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{N^{train}}$$

where $\boldsymbol{x}_i$ is an input sentence and $\boldsymbol{y}$ is its syntactic dependency structure annotation. Supervised syntactic dependency parsing involves the following two problems:

- LEARNING:
  Given a training set $\mathcal{D}^{train}$, learn parameters θ of a model f_θ that can be used to parse new sentences.
- DECODING:
  Given the learned model f_θ and a sentence $\boldsymbol{x}$, derive a syntactic dependency tree $\boldsymbol{y}$ for $\boldsymbol{x}$ according to the model f_θ.

For these problems, the following two major methods have been studied:

- *Transition-based* methods:
  Use a transition system for mapping a sentence to its dependency tree.
  - LEARNING: Learn a model for assigning a higher score to the oracle next transition at each time step than non-oracle ones.
  - DECODING: Find the highest scoring transition sequence for the input sentence.
- *Graph-based* methods:
  Assign scores to substructures of a dependency tree.
  - LEARNING: Learn a model for assigning higher scores to substructures in a correct dependency tree than incorrect ones.
  - DECODING: Find the highest scoring dependency tree for the input sentence.

Typically, transition-based methods are more computationally efficient than graph-based methods. By contrast, parsing accuracy of transition-based methods is lower than accuracy of graph-based methods. To bridge the performance gap, this thesis tackles to improve transition-based methods.

Semantic Dependencies

Semantic Dependency Representations

Semantic structures are represented in various ways. One major representation is *predicate argument structure*. Predicate argument structure encodes the semantic arguments associated with a predicate. The structure is concerned with *events*:

`Event`: `who` did `what` to `whom`, `where`, `when`, and `how`

A predicate is typically a verb and represents `what` took place. Its semantic arguments represent the participants in the event, such as `who` and `whom`, as well as further event properties, such as `where`, `when` and `how`.

Consider the example sentence of Figure 1.1: "She makes and repairs computers." Each of the blue arcs below the sentence is a *semantic dependency*. A semantic dependency holds between a predicate and its argument with a semantic role. In this example, there are two predicates: "makes" and "repairs." The predicate "makes" has two semantic arguments:

$$\langle \text{makes}, \texttt{agent}, \text{She} \rangle,\ \langle \text{makes}, \texttt{theme}, \text{computers} \rangle$$

where "She" is the "maker" (`agent`) and "computers" is the "entity made" (`theme`). Similarly, the predicate "repairs" also has two semantic arguments:

$$\langle \text{repairs}, \texttt{agent}, \text{She} \rangle,\ \langle \text{repairs}, \texttt{theme}, \text{computers} \rangle$$

where "She" is the "repairer" and "computers" is the "entity repaired."

The semantic dependency representation has an important property: *generalization of surface differences*. Figure 1.2 shows two example sentences annotated with syntactic and semantic dependencies. The two sentences have the same predicate argument structure: the word "John" plays a role of "`agent`" and the word "window" plays a role of "`theme`." However, the surface and syntactic realizations of the predicate argument structure are different between the two sentences. Thus, predicate argument structures can be regarded as a representation generalized over surface representations. This property is useful for finding semantically-equivalent sentences with different surface realizetions.

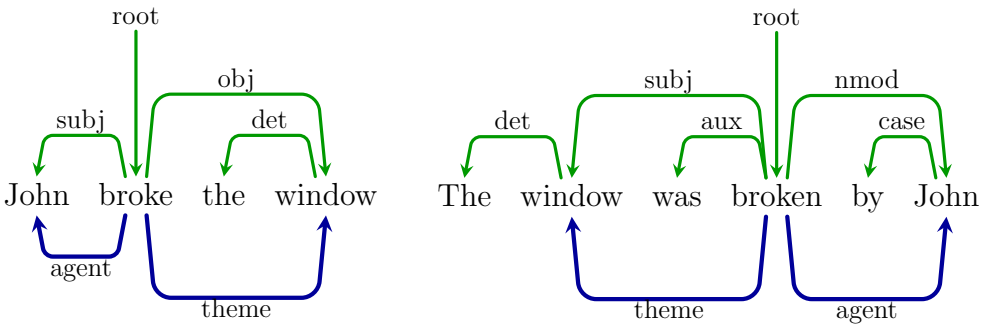


Figure 1.2: Example of two sentences that have the same predicate argument structure. Although the predicate argument structures (shown in the below part) are identical, the surface and syntactic realizations (shown in the above part) are different between the two sentences.

Semantic Dependency Parsing

A task of recovering the semantic dependencies is called *semantic dependency parsing* or *semantic role labeling* (SRL).[1] Basically, this task involves the following four steps:

- Predicate Identification:
  Identify predicates in a sentence.
- Predicate Disambiguation:
  Select a predicate sense from a set of possible senses.
- Argument Identification:
  Identify arguments of a sentence.
- Argument Classification:
  Assign a semantic role label to each argument.

Consider the example sentence in Figure 1.1. The first step would be to identify two predicates "makes" and "repairs." In the second step, for the predicate "make," the PropBank [50] defines 20 senses, $\{\text{make.01}, \cdots, \text{make.20}\}$, from

[1] In this thesis, the terms *semantic dependency parsing*, *semantic role labeling* and *predicate argument structure analysis* are used interchangeably.

which the sense used in the sentence is selected. The third step would be to identify "She" and "computers" as semantic arguments. In the fourth step, for the predicate "makes," the label "`agent`" is assigned to an argument "She" and "`theme`" to the other argument "computers."

To predict these elements, most of recent methods use supervised learning techniques. Like supervised methods for syntactic dependency parsing, supervised semantic dependency parsing methods also use a training set to learn model parameters. A basic approach learns a model for each of the four steps and uses the learned models to predict each element. Another approach combines argument identification and classification as one step and jointly predicts arguments and their labels by using a single model.

This thesis focuses on improving the one-step approach for argument identification and classification. Thus, following previous researches [39, 32], predicate identification and disambiguation is not the part of the task addressed in this thesis. In other words, given a sentence and target predicates, we predict arguments and labels.

1.2 Motivations and Problematic Issues

To identify syntactic and semantic dependencies, a variety of techniques have been proposed. However, there still remains some room for further improvement.

Feature Granularity for Syntactic Dependency Parsing

In syntactic dependency parsing, feature representations, such as surface word form and part-of-speech (POS) information, play a crucial role when predicting ambiguous dependency relationships. As features to resolve dependency ambiguities, the surface information of words is sparse while POS information is coarse. Thus, it is worthwhile to investigate intermediate representations that exist at a coarser level than the words, yet capture the information necessary to resolve dependency ambiguities.

Argument Omission in Semantic Dependency Parsing

In semantic dependency parsing, syntactic dependencies between arguments and predicates are a strong clue for identifying predicate argument structures. However, some arguments have no direct dependency relation with the predicates, which means that syntactic dependency features are not so effective for identifying such arguments. A representative example of such arguments is the ones omitted in the surface form. In particular, in pro-drop languages such as Japanese, Chinese and Italian, arguments are often omitted in text. Such *argument omission* is regarded as one of the most problematic issues facing semantic dependency parsing [29, 53, 20]. In order to overcome this problem, it is necessary to explore other types of effective features.

1.3 Solutions

Supertagging for Syntactic Dependency Parsing

To remedy the feature-granularity problem in syntactic dependency parsing, we design *supertags* based on dependency structures. Supertags, which are lexical templates extracted from dependency structure annotated corpus, encode linguistically rich information that imposes complex constraints in a local context [2]. While supertags have often been used in parsing frameworks based on lexicalized grammars, such as Lexicalized Tree-Adjoining Grammar (LTAG), Head-driven Phrase Structure Grammar (HPSG) and Combinatory Categorial Grammar (CCG), they have scarcely been utilized for dependency parsing so far. This thesis presents a framework of designing supertags specialized for dependency parsing.

Multi-Predicate Modeling for Semantic Dependency Parsing

To remedy the argument omission problem in semantic dependency parsing, we capture and exploit *multi-predicate interactions*, the relations between mul-

tiple predicates and arguments in a sentence. These relations have often been overlooked because most of existing methods attempted to solve this problem by identifying arguments per predicate without considering interactions between multiple predicates and arguments [57, 32]. However, the predicates in a sentence are semantically related to each other. Thus, exploiting this information could help to identify predicate-argument structures. In order to capture such multi-predicate interactions, this thesis presents two approaches, one using bipartite graphs and the other using grid-type recurrent neural networks.

1.4 Contributions

In summary, we make the following contributions:

- We propose a supertag design framework and develop dependency parsers exploiting various supertag sets.
- Performing experiments on Penn Treebank [40] and the Universal Dependencies [41], we demonstrate the utility of our supertags for multilingual syntactic dependency parsing.
- We propose (i) bipartite graph models which jointly identify arguments of all predicates in a sentence and (ii) grid-type recurrent neural models which automatically induce features sensitive to multi-predicate interactions exclusively from the word sequence information.
- Performing experiments on the NAIST Text Corpus [28], we demonstrate the utility of our modeling of the multi-predicate interactions for semantic dependency parsing.

1.5 Thesis Outline

The remainder of this thesis is organized as follows:

Chapter 2: Basics We provide the background knowledge to promote understanding of syntactic and semantic dependency parsing. Specifically, we describe the task settings of syntactic and semantic dependency parsing, evaluation metrics, and some methods related to our proposed methods.

Chapter 3: Syntactic Dependency Parsing We propose a framework of designing supertags. Firstly, we provide the background of supertags and formalize our framework. Then, we describe automatic assignment methods of our proposed supertags. Finally, we show experimental results of parsers exploiting supetags.

Chapter 4: Semantic Dependency Parsing We propose (i) bipartite graph models and (ii) grid-type recurrent neural models. Firstly, we provide the background of Japanese predicate argument structure analysis. Then, we describe our proposed models. Finally, we show experimental results of our models.

Chapter 5: Conclusion We summarize this thesis and discuss the future direction of the work.

Chapter 2

Preliminaries

This chapter provides the background knowledge to understand syntactic and semantic dependency parsing. Section 2.1 describes a task setting of syntactic dependency parsing, evaluation metrics and transition-based methods used in Chapter 3. Section 2.2 describes two major task settings of semantic dependency parsing, evaluation metrics and some methods related to our proposed methods in Chapter 4.

2.1 Syntactic Dependency Parsing

2.1.1 Task Definition

Given a sentence $S = w_0, w_1, \cdots, w_T$, a system predicts a syntactic dependency graph $G_S = (V_S, A_S)$:

- GIVEN: a sentence $S = w_0, w_1, \cdots, w_T$
- PREDICT: a syntactic dependency graph $G_S = (V_S, A_S)$

A sentence is a sequence of tokens[1]. This means that syntactic dependency parsing assumes that the tokenization of a sentence has already done. The first

[1]In this thesis, the terms *token* and *word* are used interchangebly.

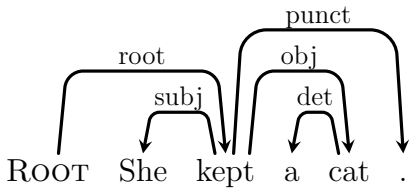


Figure 2.1: Example of a syntactic dependency graph.

token in the sentence w_0 is a special root token ROOT. A syntactic dependency graph G_S and a set of labeled arcs A_S are defined as follows:

$$V_S = \{w_0, w_1, \cdots, w_T\}$$
$$A_S \subseteq V_S \times R \times V_S,\ R = \{r_i\}_{i=1}^{M}$$

where $R = \{r_i\}_1^M$ is a set of syntactic dependency labels. A_S is a set of triples $\langle w_i, r, w_j \rangle$, where w_i is a head word, w_j is a dependent word and r is a label. A syntactic dependency graph is a *directed rooted tree*: the root node w_0 has no incoming arc (*root property*), each node has a single incoming arc (*single-head property*), and there are no cycles (*acyclicity property*) [36].

Figure 2.1 shows an example syntactic dependency graph. This graph can be denoted as follows:

$$V = \{\text{ROOT}, \text{She}, \text{kept}, \text{a}, \text{cat}, .\}$$
$$A = \{\langle \text{ROOT}, \texttt{root}, \text{kept} \rangle, \langle \text{kept}, \texttt{subj}, \text{She} \rangle,$$
$$\langle \text{kept}, \texttt{obj}, \text{cat} \rangle, \langle \text{kept}, \texttt{punct}, . \rangle, \langle \text{cat}, \texttt{det}, \text{a} \rangle\}$$

Note that, following previous work [46, 36], we assume that every word of a sentence has a position index, which makes a sentence a sequence of *unique* tokens/words. Consider the sentence:

$$\text{ROOT}_0\ \text{She}_1\ \text{likes}_2\ \text{cats}_3\ \text{and}_4\ \text{he}_5\ \text{likes}_6\ \text{docs}_7\ .{}_8$$

This sentence contains two instances of the word "likes" and we think each to be distinct from the other because their position indices are different. In this thesis, even though we explicitly denote the position indices, we assume they exist.

2.1.2 Evaluation Metrics

To evaluate syntactic dependency parsers, we follow previous researches and use the standard metrics, *attachment scores*:

- **Unlabeled Attachment Score:** The percentage of words that have the correct head.
- **Labeled Attachment Score:** The percentage of words that have the correct dependency label as well as the correct head.

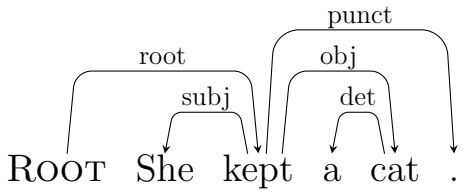
Unlabeled Attachment Score (UAS) gives an evaluation of how many head words were predicted correctly. Consider the correct dependency triple $\langle \text{kept}, \texttt{obj}, \text{cat} \rangle$ and predicted triple $\langle \text{kept}, \texttt{vmod}, \text{cat} \rangle$. Because the head "kept" for the word "cat" is identical, this triple is regarded as correct although the label is wrong. In addition to head words, Labeled Attachment Score (LAS) evaluates labels as well. Thus, the predicted triple is not regarded as correct because the label is wrong.

2.1.3 Transition-Based Methods

This section describes transition-based methods for syntactic dependency parsing. Transition-based methods are a class of data-driven dependency parsing methods exploiting machine learning techniques. In particular, this thesis focuses on *supervised* methods, which utilize sentences with correct dependency structure annotation as the input for machine learning.

In this framework, transition-based systems derive dependency trees based on a parsing model parameterized over a transition sequence from an initial to some terminal configuration. Given a training set (sentences with dependency structure annotation), a parsing model is to be induced for parsing a new sentence. Based on the induced model, a transition system, which is an abstract machine consisting of a set of *configurations* and *transitions* between configurations, derives the optimal dependency tree [36].

This approach was pioneered by Kudo and Matsumoto [37], Yamada and Matsumoto [59], and Nivre [47] for unlabeled dependency parsing. Nivre et al. [48] and Nivre and Scholz [49] extended the approach to labeled dependency parsing.



Step	Transition	Stack	Buffer	A
1	SHIFT	[ROOT]	[She kept a cat .]	$\emptyset$
2	SHIFT	[ROOT She]	[kept a cat .]	
3	SHIFT	[ROOT She kept]	[a cat .]	
4	LEFT-ARC(subj)	[ROOT kept]	[a cat .]	$A \cup$ (kept, She)
5	SHIFT	[ROOT kept a]	[cat .]	
6	SHIFT	[ROOT kept a cat]	[.]	
7	LEFT-ARC(det)	[ROOT kept cat]	[.]	$A \cup$ (cat, a)
8	RIGHT-ARC(obj)	[ROOT kept]	[.]	$A \cup$ (kept, cat)
9	SHIFT	[ROOT kept .]	[]	
10	LEFT-ARC(punct)	[ROOT kept]	[]	$A \cup$ (kept, .)
11	LEFT-ARC(root)	[ROOT]	[]	$A \cup$ (ROOT, kept)

Figure 2.2: Example of a parsing process with the *arc-standard model*.

Instead of the greedy search used in the previous systems, beam search was applied to dependency parsing by Zhang and Clark [63, 64]. Of the variations of transition-based systems, *arc-standard* and *arc-eager* are representative systems, and the implementation MaltParser has been widely used so far [46]. In this thesis, we employ the arc-standard model [45].

In the arc-standard model, the configuration $c = (s, b, A)$ consists of a *stack* s, *buffer* b, and set of *dependency arcs* A. The initial configuration for an input sentence $S = w_0, w_1, \cdots, w_T$ is $s = [\texttt{ROOT}]$, $b = [w_1, \cdots, w_T]$, and $A = \emptyset$. A configuration c is terminal if the buffer is empty and the stack contains the single node ROOT, and the parse tree is given by A_c. Denoting s_i $(i = 1, 2, \cdots)$ as the i_{th} word on the top of the stack, and b_j $(j = 1, 2, \cdots)$ as the j_{th} element on the buffer, the arc-standard system defines the following three types of transitions:

- `LEFT-ARC`: adds an arc $s_1 \rightarrow s_2$ and removes s_2 from the stack under the precondition $\mid s \mid \geq 2$.
- `RIGHT-ARC`: adds an arc $s_2 \rightarrow s_1$ and removes s_1 from the stack under the precondition $\mid s \mid \geq 2$.
- `SHIFT`: moves b_1 from the buffer to the stack under the precondition $\mid b \mid \geq 1$.

Consider the sentence "She kept a cat ." in Figure 2.2. At step 6, `LEFT-ARC` is chosen as the next transition, so that the second top word on the stack "`a`" (s_2) depends on the top word "cat" (s_1) and is removed from the stack. At step 7, `RIGHT-ARC` is chosen as the next transition; hence, the top word on the stack "cat" (s_1) depends on the second top word "kept" (s_2) and is removed from the stack. At step 8, `SHIFT` is chosen as the next transition, and the first word in the buffer "." (b_1) is removed from the buffer and moved to the stack.

As a result of such transitions, the goal of a transition-based system is to predict a correct transition sequence based on each configuration. Specifically, the system chooses the most probable next transition at each configuration based on *scores*. The scores are computed as the dot product of the *weight* vector and $feature$ vector. The feature vector is made by predefined features. The features are represented using some lexical information based on the current configuration, such as the word forms and POS tags of some words on the stack/buffer.

2.2 Semantic Dependency Parsing

2.2.1 Task Definition

A task of recovering the predicate argument structure of a sentence is called *semantic dependency parsing* or *semantic role labeling* (SRL). There are two major task settings tackled by many researchers: one is adopted in the CoNLL-2008 shared task [56] and the other is in the CoNLL-2009 shared task [19]. This section describes the two task settings, in particular, identification of predicate argument relations.[2]

[2]Predicate sense disambiguation is out of scope of this thesis because we focus on identification of predicate argument structures.

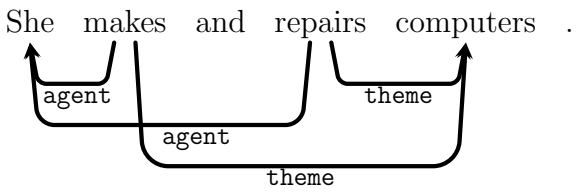


Figure 2.3: Example of semantic dependencies.

The CoNLL-2008 Shared Task

Given a sentence $S = w_1, \cdots, w_T$, a system identifies predicate argument relations $Y = \{\langle p, r, a\rangle_i\}_1^N$.

- GIVEN: a sentence $S = w_1, \cdots, w_T$
- PREDICT: a set of predicate argument relations $Y = \{\langle p, r, a\rangle_i\}_{i=1}^N$

where each triple consists of a predicate $p \in V_S^{prd}$, its argument $a \in V_S^{arg}$ and a label $r \in R$:

$$V_S^{prd} = \{p_1, \cdots, p_M\} \subseteq S$$
$$V_S^{arg} = \{a_1, \cdots, a_N\} \subseteq S$$
$$R = \{r_i\}_{i=1}^K$$

Thus, a predicate argument triple is defined in the following space:

$$\langle p, r, a\rangle \in V_S^{prd} \times R \times V_S^{arg}$$

Figure 2.3 illustrates an example. There are two predicates: "makes" and "repairs." The predicate "makes" has two arguments: one is "She" with the agentive role and the other is "computers" with the thematic role. The other predicate "repairs" also has the same arguments. To sum up, a system is expected to return the following triples for this sentence:

$$\{\ \langle \text{makes}, \texttt{agent}, \text{She}\rangle, \langle \text{makes}, \texttt{theme}, \text{computers}\rangle,$$
$$\langle \text{repairs}, \texttt{agent}, \text{She}\rangle, \langle \text{repairs}, \texttt{theme}, \text{computers}\rangle\ \}$$

The CoNLL-2009 Shared Task
The main difference between the CoNLL-2008 and CoNLL-2009 shared tasks is whether or not to provide target predicates. Specifically, in the CoNLL-2009 shared tasks, given a sentence $S = w_1, \cdots, w_T$ and the target predicates $V_S^{prd} = \{p_1, \cdots, p_M\}$, a system identifies arguments a with their labels r for each predicate p. In other words, predicates do not have to be identified. This setting is also adopted in Japanese predicate argument structure analysis. Thus, the task setting in Chapter 4 assumes that target predicates are given and predicate identification is not the part of the task.

2.2.2 Evaluation Metrics

To evaluate semantic dependency parsers, the most widely used metric is the *F1 score*, which is the harmonic mean of precision and recall:

- **Precision:** The percentage of triples $\langle w_p, r, w_a \rangle$ in the system output that were correct.
- **Recall:** The percentage of triples $\langle w_p, r, w_a \rangle$ annotated in a dataset that were correctly identified.
- **F1 score:** $\dfrac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$

For example, a system predicted 5 triples for a sentence and a target predicate, and if 3 out of the 5 triples were correct, precision is $3/5 = 0.60$. Similarly, if 3 out of 4 triples annotated in a dataset were correctly identified by the system, recall is $3/4 = 0.75$. Thus, F1 score is calculated as $\frac{2 \times 0.60 \times 0.75}{0.60 + 0.75} = 0.67$.

2.2.3 Related Methods

To promote understanding of our proposed methods in Chapter 4, this section describes some related methods.

A Basic Pipeline Proceudure

Identification of predicate argument relations involves the following tree steps:

1. Predicate Identification:
   Identify predicates in a sentence.

2. Argument Identification:
   Identify the arguments of a predicate.

3. Argument Classification:
   Assign a semantic role label to each argument of a predicate.

The first step is to identify predicates in a sentence $S = w_1, \cdots, w_T$:

$$V_S^{prd} = f^{pi}(S)$$

where a predicate identification model f^{pi} returns a set of predicates in a sentence V_S^{prd}. For example, the sentence in Figure 2.3 contains the following two predicates: "makes" and "repairs." Thus, the model f^{pi} is expected to return the following set:

$$V_S^{prd} = \{\text{makes}, \text{repairs}\}$$

Then, the second step is to identify semantic arguments of each predicate $p \in V^{prd}$:

$$V_p^{arg} = f^{ai}(p, S)$$

For example, for the predicate "makes" in Figure 2.3, the model f^{ai} is expected to return the following set:

$$V_{\text{makes}}^{arg} = \{\text{She}, \text{computers}\}$$

Finally, the third step is to assign a semantic role label to each pair of a predicate $p \in V_S^{prd}$ and its argument $a \in V_p^{arg}$:

$$r = f^{ac}(p, a, S)$$

For example, for the predicate "makes" and its argument "She" in Figure 2.3, the model f^{ac} is expected to return the following label:

$$r = \texttt{agent}$$

Now we can construct a predicate argument triple:

$$\langle p, r, a \rangle = \langle \text{makes}, \texttt{agent}, \text{She} \rangle$$

Other predicate argument triples are predicted in the same manner.

The Top English System in the CoNLL-2009 Shared Task

A variety of methods for semantic dependency parsing have been proposed. Among them, we describe methods used for the top system [4] that achieved the best result for English in the CoNLL-2009 shared task[3] [19]. In this task, given a sentence and the target predicates, systems predict predicate argument triples $\{\langle p, r, a \rangle_k\}_1^N$. The top system adopts a pipeline procedure, which firstly identifies arguments (Argument Identification) and then assigns a semantic role label for each argument (Argument Classification).

For argument identification, they use a binary classifier to calculate the probability that a word in the sentence is an argument. Given a sentence $S = w_1, \cdots, w_T$ and the target predicates $V_S^{prd} = \{p_i\}_1^N$, the classifier f^{ai} calculate the probability:

$$\mathrm{P}(w_t = \texttt{arg} | S, V_S^{prd}) = f^{ai}(w_t, S, V_S^{prd}) = \mathrm{sigmoid}(\mathbf{w} \cdot \boldsymbol{\phi}^{ai}(w_t, S, V_S^{prd}))$$

where $\mathbf{w}$ is a weight vector and $\boldsymbol{\phi}(w_t, S)$ is a feature vector. If the probability is greater than 0.5, the system determines that the word is an argument.

For argument classification, they use a multiclass classifier. Each class r corresponds to a certain label. Given a sentence $S = w_1, \cdots, w_T$, a predicate p and an argument a, the classifier f^{ac} calculate the probability:

$$\mathrm{P}(r | S, p, a) = f^{ac}(r, S, p, a) = \frac{\exp(\mathbf{w} \cdot \boldsymbol{\phi}^{ac}(r, S, p, a))}{\sum_{r' \in R} \exp(\mathbf{w} \cdot \boldsymbol{\phi}^{ac}(r', S, p, a))}$$

The class with the highest probability $\hat{r}$ is selected as the resulting label:

$$\hat{r} = \operatorname*{argmax}_{r \in R} \mathrm{P}(r | S, p, a)$$

[3]Closed Challenge, SRL-only Task, Semantic Labeled F1.

By predicting a label $\hat{r}$ for each pair of a predicate $p \in V_S^{prd}$ and its argument $a \in V_S^{arg}$, we can obtain a semantic dependency triple $\langle p, r, a \rangle$.

A Basic Method for Japanese PAS Analysis

One of the methods employed for Japanese predicate argument structure analysis is a pointwise method [32], which is used as a baseline in Chapter 4. This method selects the most probable argument a for a predicate p and label r:

$$\langle p, r, \hat{a} \rangle = \underset{a \in V_S^{arg}}{\operatorname{argmax}} \mathrm{P}(a|S, r, p)$$

where the argument with the highest probability $\hat{a}$ is selected from a set of candidate arguments $V_S^{arg} = S \cup \{\textsc{None}\}$. The special argument None is expected to be selected when the predicate p has no argument with a label r. The probability $\mathrm{P}(a|S, r, p)$ is defined as follows:

$$\mathrm{P}(a|S, r, p) = \frac{\exp(\mathbf{w} \cdot \boldsymbol{\phi}(a, r, S, p))}{\sum_{a' \in V_S^{arg}} \exp(\mathbf{w} \cdot \boldsymbol{\phi}(a', r, S, p))}$$

This simple method can achieve high-performance depending on feature engineering. In particular, syntactic features, related to syntactic dependency relations and POS tags, are a key to high-performance. Recent methods, however, yield good results without such features by using neural networks.

A State-of-the-Art Method Using RNNs

Zhou et al. [67] achieved the state-of-the-art result without syntactic information in the English SRL task. Our proposed method in Section 4.4 has been inspired by their method.

Their method uses stacked bidirectional RNNs (Bi-RNN) [54, 16, 17]. The overall architecture consists of the following three components:

- **Input Layer**: Map each word to a feature vector representation.
- **RNN Layer**: Produce high-level feature vectors using Bi-RNNs.
- **Output Layer**: Compute the probability of each label for each word using the softmax function.

Given an input sentence $S = w_1, \cdots, w_{\mathrm{T}}$ and a target predicate p, the input layer maps each word w_t to a d_x-dimensional vector $\mathbf{x}_t \in \mathbb{R}^{d_x}$. This vector representation is constructed by concatenating the following four types of vectors:

$$\mathbf{x}_t = \mathbf{x}_t^{arg} \oplus \mathbf{x}_t^{pred} \oplus \mathbf{x}_t^{ctx} \oplus \mathbf{x}_t^{mark} \tag{2.1}$$

Each of the three vectors is based on the following atomic features:

`ARG:` Word index of each word.

`PRED:` Word index of the target predicate.

`CTX:` Word index of the target predicate and the C words around the predicate.

`MARK:` Binary index that represents whether or not the word is included in `CTX-P`.

For the `ARG` feature, a word index x^{word} for each word w is extracted from a set of word indices $\mathcal{V}$. Similarly, for the `PRED` feature, we extract word index x^{word} for the predicate p. The `CTX` feature consists of each word index for the C words taking the target predicate at the center, where C denotes the window size. The `MARK` feature is a binary value $\{0, 1\}$ that represents whether or not the word is included in the C context words.

Then, using these feature indices, each feature vector is looked up from each embedding matrix. Each embedding matrix stores column vectors, each of which corresponds to each feature index. For example, an argument feature vector $\mathbf{x}^{arg}$ is looked up from a word embedding matrix $\mathbf{E}^{arg} \in \mathbb{R}^{d^{arg} \times |\mathcal{V}|}$. The resulting vectors are concatenated as a feature vector $\mathbf{x}_t$ (Eq. 2.1).

Each feature vector $\mathbf{x}_t$ is multiplied with a parameter matrix $\mathbf{W}_x$:

$$\mathbf{h}_t^{(0)} = \mathbf{W}_x \mathbf{x}_t$$

The vector $\mathbf{h}_t^{(0)}$ is given to the first RNN layer as input.

In the RNN layers, feature vectors are updated recurrently using Bi-RNNs. Bi-RNNs process an input sequence in a left-to-right manner for odd-numbered layers and in a right-to-left manner for even-numbered layers. By stacking these layers, we can construct the deeper network structures.

Stacked Bi-RNNs consist of L layers, and the hidden state in the layer $\ell \in (1, \cdots, L)$ is calculated as follows:

$$\mathbf{h}_t^{(\ell)} = \begin{cases} g^{(\ell)}(\mathbf{h}_t^{(\ell-1)}, \mathbf{h}_{t-1}^{(\ell)}) & (\ell = \text{odd}) \\ g^{(\ell)}(\mathbf{h}_t^{(\ell-1)}, \mathbf{h}_{t+1}^{(\ell)}) & (\ell = \text{even}) \end{cases}$$

Both of the odd- and even-numbered layers receive $\mathbf{h}_t^{(\ell-1)}$, the t-th hidden state of the $\ell - 1$ layer, as the first input of the function $g^{(\ell)}$, which is an arbitrary function. For the second input of $g^{(\ell)}$, odd-numbered layers receive $\mathbf{h}_{t-1}^{(\ell)}$, whereas even-numbered layers receive $\mathbf{h}_{t+1}^{(\ell)}$. By calculating the hidden states until the L-th layer, we obtain a hidden state sequence $\mathbf{h}_{1:T}^{(L)} = (\mathbf{h}_1^{(L)}, \cdots, \mathbf{h}_T^{(L)})$. Using each vector $\mathbf{h}_t^{(L)}$, we calculate the probability of labels for each word in the output layer.

In the output layer, a label sequence probability $y_{1:T}$ is calculated using conditional random fields (CRF) [38]:

$$\mathrm{P}(y_{1:T}|\mathbf{h}_{1:T}^{(L)}) = \mathrm{CRF}(\mathbf{h}_{1:T}^{(L)}, y_{1:T})$$

where $\mathbf{h}_{1:T}^{(L)}$ is a sequence of vector representations propagated from the last RNN layer. Each element $\mathbf{y}_t$ of $y_{1:T}$ is a label for a word w_t. The label sequence with the maximum probability is output as a result.

A variant of this method has been proposed by He et al. [23]. They use the same architecture as the one mentioned above but simplify the features. By using sophisticated learning methods (e.g. the RNN dropout and orthogonal initialization), their method achieves the state-of-the-art results.

Chapter 3

Syntactic Dependency Parsing: Supertag Design Framework

3.1 Introduction

Data-driven dependency parsing approaches, which make use of machine learning, have achieved great success in the automatic syntactic analysis of natural language [36]. In data-driven approaches, *transition-based* dependency parsing, which utilizes a deterministic shift-reduce process for structural prediction, has received considerable attention because of its low time complexity and the freedom to design features based on a rich context [66]. In particular, the feature definition is the key to the high performance of transition-based dependency parsers.

As feature representations, lexical information, including surface word form and part-of-speech (POS) information, plays a crucial role when predicting ambiguous dependency relationships. However, as features to resolve dependency ambiguities, the surface information of words is sparse while POS information is coarse. Therefore, it is worthwhile to investigate intermediate representations that exist at a coarser level than the words, yet capture the information necessary to resolve dependency ambiguities [35].

To improve syntactic dependency parsing, we focus on defining *supertags*. Supertags are tags extended from the notion of POS tags and represent rich syntactic information [43], such as the head direction and dependency label. Figure 3.1 illustrates an example of our dependency-based supertags. In this example, each supertag encodes the head direction with the dependency label and dependent direction.

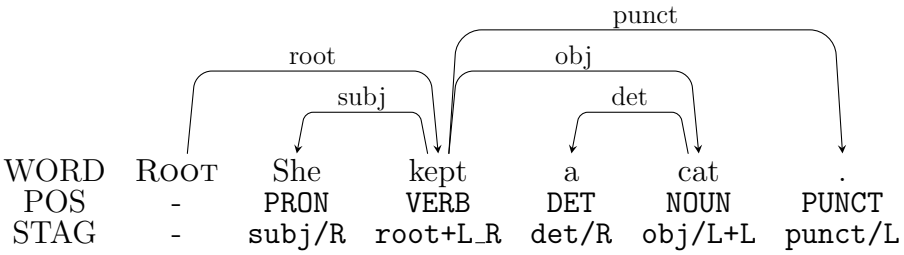


Figure 3.1: Illustrative example of supertags for the dependency structure. Supertags encode syntactic information, e.g., the head direction and dependency label.

While supertags can arbitrarily be designed, it is important to keep the adequate balance between the supertag granularity and predictability to improve parsing performance. Increasing the granularity of supertags to capture more fine-grained syntactic information results in large tag sets, which tend to be more difficult to predict automatically. To improve dependency parsing performance by utilizing supertags, it is necessary to design supertags that have the following two properties: (i) easy to be automatically assigned to the sentence and (ii) expressive enough to resolve dependency ambiguities.

In this chapter, we present a supertag design framework that allows us to design supertag sets at various granularity levels. First, we formalize the supertag design framework and instantiate various granularity-level supertag sets. To investigate the appropriate granularity or design of supertags needed to improve parsing performance, we build various granularity-level supertag sets based on the framework. Then, using the supertag sets as features, we perform experiments on multilingual dependency parsing. For English dependency parsing with the supertags, we perform experiments on the Penn Treebank data set. In addition, the utility of the supertags for multilingual dependency parsing is an open question, so we also perform experiments on Universal Dependencies data set (UD; release 1.3).[1] The experimental results show that appropriately designed supertags are effective for dependency parsing.

In summary, the main contributions of this chapter are as follows.

[1] http://universaldependencies.org/

1. We present a supertag design framework.

2. We develop transition-based dependency parsers exploiting various supertag sets.

3. We demonstrate the utility of our supertags for multilingual dependency parsing and suggest which syntactic clues should be incorporated into supertags.

3.2 Supertag Design Framework

The main challenge when designing supertags is to find the right balance between granularity and predictability. Ideally, we would like to increase the granularity of the supertags to capture finer-grained syntactic information, but large tag sets tend to be more difficult to predict automatically. This section provides a supertag design framework.

3.2.1 Supertag Design Framework

Figure 3.1 shows an example sentence and its supertags. Supertags are based on head and dependent information to capture local syntactic context. Thus, we assume that there is a set of labeled directed arcs A_S for a sentence $S = w_0, w_1, \cdots, w_T$. The labeled directed arcs of the example sentence is denoted as follows:

$$A = \{\langle \textsc{Root}, \texttt{root}, \text{kept}\rangle, \langle \text{kept}, \texttt{subj}, \text{She}\rangle, \\ \langle \text{kept}, \texttt{obj}, \text{cat}\rangle, \langle \text{kept}, \texttt{punct}, .\rangle, \langle \text{cat}, \texttt{det}, \text{a}\rangle\}$$

where a triple $\langle w_i, r, w_j\rangle$ represents a dependency relation from head w_i to dependent w_j labeled with relation type r. Using such syntactic information, we design supertag sets at various granularity levels.

Specifically, a supertag of a word $w.stag$ is defined as follows:

$$w.stag = \textsc{Stag}(w) \tag{3.1}$$

where the function $\textsc{Stag}(w)$ can arbitrarily be defined and returns a supertag. In this thesis, we define the function based on the syntactic information of the head and dependents:

$$\textsc{Stag}(w) = \langle \textsc{Head}(y_w), \textsc{Dep}(D_w) \rangle \tag{3.2}$$

where the argument y_w of the function $\textsc{Head}(\cdot)$ is a dependency triple:

$$y_w = \langle h, r, w \rangle \in A$$

Based on the triple, the function $\textsc{Head}(\cdot)$ returns the head-related information. Also, the argument D_w of the function $\textsc{Dep}(\cdot)$ in Equation 3.2 is a set of dependency triples that have w as a head:

$$D_w = \{\langle h, r, d \rangle \in A | h = w\}$$

Based on this set, the function $\textsc{Dep}(\cdot)$ returns the dependent-related information. The variability of syntactic granularity can be represented by the definitions of $\textsc{Head}(y_w)$ and $\textsc{Dep}(D_w)$.

3.2.2 Supertag Instantiation

Using the above-mentioned generic supertag design framework, we instantiate a variety of supertag sets. As a basic instantiation, we define the functions in Equation 3.2 using the information of *dependency labels* and *head directions* between a word and its head/dependents.

As the head information, we define the function $\textsc{Head}(y_w)$ as follows:

$$\textsc{Head}(y_w) = \langle \textsc{HeadLabel}(y_w), \textsc{HeadDir}(y_w) \rangle \tag{3.3}$$

where $\textsc{HeadLabel}(y_w)$ returns the dependency label r from the triple $y_w = \langle h, r, w \rangle$. $\textsc{HeadDir}(y_w)$ returns the direction of the head word h relative to the target word w. That is, this function returns either of the three values: left (L), right (R) or Null. If a word w has "Root" as its head, we consider it as having no direction, so Null is returned.

In addition to the head information, we add dependent information by defining the function $\textsc{Dep}(D_w)$ as follows:

$$\begin{aligned} \textsc{Dep}(D_w) = \langle & \textsc{hasDep}(D_w, L), \textsc{DepLabel}(D_w, L), \\ & \textsc{hasDep}(D_w, R), \textsc{DepLabel}(D_w, R) \rangle \end{aligned} \tag{3.4}$$

	HeadLabel	HeadDir	hasDep	DepLabel
Stag-A	✓	✓	✓	✓
Stag-B	✓	✓	✓	×
Stag-C	✓	✓	×	✓
Stag-D	✓	×	×	✓
Stag-E	✓	✓	×	×
Stag-F	×	✓	✓	×
Stag-G	✓	×	×	×

Table 3.1: Various granularity supertag sets. The mark ✓ indicates the supertag set is defined using the function, and the mark × indicates not.

where hasDep$(D_w, L/R)$ returns True$_{L/R}$ if a word has any left (L) or right (R) dependents; otherwise, it returns False. DepLabel$(D_w, L/R)$ returns a set of the dependency labels of obligatory left/right dependents. In this thesis, we define obligatory dependents as the ones that have one of the following dependency relation labels: "`SUB`," "`OBJ`," "`PRD`," or "`VC`" in the Penn Treebank [40], and "`nsubj`," "`nsubjpass`," "`dobj`," "`iobj`," "`csubj`," "`csubjpass`," or "`ccomp`" in Universal Dependencies [41].

In previous work on supertag design, Foth et al. [14] defined Dep$(\cdot)$ as the function that encodes the order of dependents as well as the dependent labels. However, we do not consider the order to avoid increasing the number of tags.

Based on Equations 3.3 and 3.4, we define various granularity-level supertag sets by ablating each function. Table 3.1 shows our seven supertag sets. Stag-A is the most basic instantiation using all the functions. Ablating the function DepLabel from Stag-A, we can instantiate Stag-B, which encodes no dependency labels for obligatory dependents. Similarly, other supertag sets are instantiated by ablating other functions.

3.2.3 Supertag Notation

Supertag notations for each supertag set can be defined arbitrarily. As an example, we introduce our notations for each supertag set, shown in Table 3.2.

	She	kept	a	cat	.
STAG-A	subj/R	root+subj/L_obj/R	det/R	obj/L+L	punct/L
STAG-B	subj/R	root+L_R	det/R	obj/L+L	punct/L
STAG-C	subj/R	root+subj_obj	det/R	obj/L	punct/L
STAG-D	subj	root+subj_obj	det	obj	punct
STAG-E	subj/R	root	det/R	obj/L	punct/L
STAG-F	R	L_R	R	L	L
STAG-G	subj	root	det	obj	punct

Table 3.2: Examples of supertags for the sentence "She kept a cat .".

Consider the word "kept" in the example sentence in Figure 2.2. In the supertag set STAG-A, the word "kept" is assigned the following supertag:

$$\text{kept}.stag = \texttt{root+subj/L\_obj/R}$$

where the part before "`+`" specifies the head information ("`root`") and the part afterwards specifies the dependent information ("`subj/L_obj/R`").

To create this tag, we first encode the head information of "kept" using the functions in Equation 3.3:

$$\textsc{HeadLabel}(y_{\text{kept}} = \langle \textsc{Root}, \texttt{root}, \text{kept}\rangle) = \texttt{root}$$
$$\textsc{HeadDir}(y_{\text{kept}} = \langle \textsc{Root}, \texttt{root}, \text{kept}\rangle) = \textsc{Null}$$

where the function $\textsc{HeadLabel}(\cdot)$ returns the dependency label "`root`" and the function $\textsc{HeadDir}(\cdot)$ returns the head direction "NULL." As a result, we obtain the following head information:

$$\textsc{Head}(y_{\text{kept}} = \langle \textsc{Root}, \texttt{root}, \text{kept}\rangle) = \langle \texttt{root}, \textsc{Null}\rangle$$

where the head information $\langle \texttt{root}, \textsc{Null}\rangle$ is converted into the supertag notation as "`root`," in which "NULL" is not literally specified. Note that if the direction is "L" (or "R"), we convert it as "`root/L`" (or "`root/R`").

We then encode dependent information using the function $\textsc{Dep}(D_w)$ in Equation 3.4. The argument of the function for the word "kept" is as follows:

$$D_{kept} = \{\langle \text{kept}, \texttt{subj}, \text{She}\rangle, \langle \text{kept}, \texttt{obj}, \text{cat}\rangle, \langle \text{kept}, \texttt{punct}, .\rangle\}$$

where D_{kept} is a set of triples that consists of the head "kept" and its dependents with labels. Based on this set, the following dependent information is calculated:

$$\begin{aligned}
&\textsc{hasDep}(D_{kept}, L) = \textsc{True}_L \\
&\textsc{hasDep}(D_{kept}, R) = \textsc{True}_R \\
&\textsc{DepLabel}(D_{kept}, L) = \{\texttt{subj}\} \\
&\textsc{DepLabel}(D_{kept}, R) = \{\texttt{obj}\}
\end{aligned}$$

where $\textsc{hasDep}(\cdot)$ returns a boolean variable, i.e., $\textsc{True}_{L/R}$ or $\textsc{False}_{L/R}$. Also, $\textsc{DepLabel}(\cdot)$ returns a set of labels of all the left/right dependents.[2] As a result, we obtain the following dependent information:

$$\textsc{Dep}(D_{\text{kept}}) = \langle \textsc{True}_L, \{\texttt{subj}\}, \textsc{True}_R, \{\texttt{obj}\} \rangle$$

We convert this information into the supertag notation as "`subj/L_obj/R`."

Finally, concatenating the obtained head and dependent information, we obtain the following supertag:

`root+subj/L_obj/R`

where "`+`" indicates the boundary of the head and dependent information. Depending on each supertag set, different syntactic information is ablated from Stag-A and encoded for each supertag, as shown in Table 3.2.

3.3 Dependency Parsers Exploiting Supertags

To exploit supertags in transition-based dependency parsing systems, we need to automatically assign them to each word. We conduct the automatic assignment of our designed supertags by adopting the same approach used in sequence labeling tasks such as POS tagging.

3.3.1 Automatic Supertag Assignment

We build a supertagger to assign a supertag $u \in U$ to the word w_t in a sentence $S = w_1, \cdots, w_T$ in a left-to-right manner, based on the following equation:

$$\hat{u} = \underset{u \in U}{\text{argmax}} \ \ \mathbf{w} \cdot \phi(w_i, u, S) \tag{3.5}$$

[2]For example, if the target word has two left dependents with the labels "`dobj`" and "`iobj`," $\textsc{DepLabel}(\cdot)$ returns $\{\texttt{dobj}, \texttt{iobj}\}$.

where the score of supertag u is computed by the dot-product of the weight vector $\mathbf{w}$ and feature vector ϕ. The highest scoring supertag $\hat{u}$ is picked for the target word w_i. The weight vector $\mathbf{w}$ is trained on a training set. The feature vector ϕ is defined by using feature templates. We extract features from a 7-word window (the *feature window*) surrounding the target word w_i using the feature templates shown in Table A.1.

We define UNIGRAM, BIGRAM, and HISTORY feature templates. As the UNIGRAM feature templates, we use the surface word form and POS tag of each word in the feature window. As theBIGRAM feature templates, we define conjunctive features by concatenating the surface word form and POS tag information of some specific pairs of the words in the feature window. In addition to these feature templates, we dynamically utilize the supertags, which have already been predicted during the tagging process, as features. When assigning a supertag to the target word w_i in the feature window, the previous words, such as w_{i-1} and w_{i-2}, have already been assigned a supertag, which is expected to be helpful for predicting the supertag of the target word. Hence, we define the HISTORY feature templates by combining those supertags assigned to w_{i-1} and w_{i-2} with the POS tags or surface word forms.

A supertagging model instantiates the features from those feature templates. Using the supertags automatically predicted by the supertagging model, we conduct dependency parsing.

3.3.2 Supertag Features for Dependency Parsing

We employ the *arc-standard* model as a transition-based dependency parsing system. Specifically, the system chooses the highest scoring next transition $\hat{t}$ at each configuration c based on the following equation:

$$\hat{t} = \underset{t \in T}{\operatorname{argmax}} \ \mathbf{w} \cdot \phi(t, c) \tag{3.6}$$

where the score of transition t is computed by the dot-product of the weight vector $\mathbf{w}$ and feature vector ϕ. The highest scoring transition $\hat{t}$ of the possible transition set T is picked up as the next transition. The weight vector is trained on a training set. The feature vector is defined with the feature templates shown in Table A.2. Each feature template is defined with information drawn from the

feature window, which consists of the top three words (or partial structures) on the stack and the first three words on the buffer.

UNIGRAM, BIGRAM, and STRUCTURAL features are based on the features used in [15] and [26] with some modifications, which we call *base features*. By contrast, UNISTAG and BISTAG are new feature templates related to supertags, which we call *supertag features*.

For the UNISTAG features ($\langle p.stag \rangle$), we use the supertag of each word p within the feature window. To consider a broader context, we define the BISTAG features. For some specific pairs (p, q) of the words within the feature window, we set the conjunctive features as BISTAG, such as conjunction of the two supertags ($\langle p.stag \circ q.stag \rangle$). To investigate the utility of these supertag features, we perform experiments.

3.4 Experiment

This section presents experiments and results for supertagging and transition-based dependency parsing exploiting supertags.

3.4.1 Datasets

We performed experiments on English dependency parsing and multilingual dependency parsing.

English Dependency Parsing

For English dependency parsing, we performed experiments on the Wall Street Journal part of the Penn Treebank (PTB) dataset [40]. We converted the constituent trees into two types of dependency format:

- Yamada and Matsumoto head rules (PTB-YM) [59] using Penn2Malt[3]
- Stanford dependencies (PTB-SD) [12] using the converter[4]

We adopted the standard splits, using sections 2-21 for training, section 22 for development, and section 23 for testing. We assigned POS tags to the training

[3]http://stp.lingfil.uu.se/ nivre/research/Penn2Malt.jar

[4]http://nlp.stanford.edu/software/stanford-dependencies.shtml

data by ten-fold jackknifing, following [26]. The development and test sets were automatically tagged by the POS-tagger trained on the training set.

Multilingual Dependency Parsing
For multilingual dependency parsing, we used the Universal Dependencies (UD; release 1.3) data set [41]. This data set has cross-linguistically consistent treebank annotation for many languages. The annotation scheme is an extension of the Stanford dependencies [12, 13, 11], Google universal part-of-speech tags [52], and the Interset interlingua for morphosyntactic tagsets [62].

For the target languages, we chose six languages from different language branches:

- Arabic (AR) from the Semitic languages
- German (DE) from the Germanic Languages
- Spanish (ES) from the Italic languages
- Indonesian (ID) from the Malayo-Polynesian languages
- Russian (RU) from the Slavic languages
- Chinese (ZH) from the Sinitic languages

In some language data sets, there are no fine-grained POS tags. In that case, we used the coarse-grained ones.

3.4.2 Setup of Supertagging Experiments

To train supertagging models, we used the averaged perceptron [9] with max violation updates [25]. The number of iterations was to 10. For decoding, we exploited beam search with a beam width of 8. Tables 3.3 and 3.4 show the size of each supertag set in the Penn Treebank and Universal Dependencies, respectively.

3.4.3 Setup of Parsing Experiments

To train parsing models, we used the averaged perceptron with max violation updates in the same manner as the supertagging experiments. The number of

	PTB-YM	PTB-SD
STAG-A	**321**	**896**
STAG-B	79	231
STAG-C	165	528
STAG-D	127	412
STAG-E	21	85
STAG-F	12	12
STAG-G	12	49

Table 3.3: Supertag statistics on Penn Treebank. "PTB-YM" is the Yamada & Matsumoto dependency scheme [59], and "PTB-SD" is the Stanford dependency scheme [12].

	UD-AR	UD-DE	UD-ES	UD-ID	UD-RU	UD-ZH	UD-Avg.
STAG-A	**998**	**916**	**1209**	**680**	**655**	**630**	**847.50**
STAG-B	183	227	216	186	233	175	203.33
STAG-C	522	431	558	370	306	360	424.50
STAG-D	442	321	445	305	237	300	341.67
STAG-E	57	64	61	55	74	64	62.50
STAG-F	12	11	11	11	11	11	11.17
STAG-G	31	33	32	30	39	38	33.83

Table 3.4: Supertag statistics on Universal Dependencies [41].

iterations is set to 20. For decoding, we exploited beam search with a beam width of 16. To evaluate the utility of the supertags for *arc-standard* dependency parsers, we used the parsers without supertags as the baseline and compared them with the parsers with supertags. The supertags used for the parsers were automatically predicted. Following the same procedure as automatic POS tagging, we assigned the proposed supertags to the training data by ten-fold jackknifing. For the development and test set, we automatically assigned the supertags using a supertagger trained on the whole training set.

	PTB-YM	PTB-SD
STAG-A	88.05	87.75
STAG-B	88.94	88.92
STAG-C	89.71	89.67
STAG-D	90.77	90.84
STAG-E	90.56	90.71
STAG-F	**91.61**	91.81
STAG-G	91.60	**91.86**

Table 3.5: Supertagging results on the English Penn Treebank. Each number indicates accuracy.

3.4.4 Results for Suppertagging

Tables 3.5 and 3.6 show the accuracies of automatic supertagging in English and other languages, respectively. These results suggest that what kind of syntactic information is easy or difficult to predict as sequential labeling.

In English, the results of STAG-F or STAG-G have the highest accuracy. This is consistent with the results in other languages in UD. As Table 3.1 shows, STAG-F encodes the head directionality and left/right dependent possession information, and STAG-G encodes the dependency label on the edge between the target word and its head. Generally, because smaller tag sets tend to be easier to predict than larger ones, the accuracies of the two supertag sets are higher than others. However, in the six languages of UD, although STAG-F is smaller than STAG-G, the average STAG-G accuracy (88.44% on average for the six UD languages) is higher than the average STAG-F accuracy (87.35%). This suggests that it is not always difficult to predict larger tag sets and, furthermore, the prediction complexity changes according to what information the target tag set encodes. The results for STAG-F suggest that the syntactic information encoded by STAG-F is more difficult to predict as sequential labeling task than the head dependency labels encoded by STAG-G in the majority of languages.

Similarly, regardless of the tag set size, the prediction accuracy of STAG-B (82.45/88.94/88.92% for UD-Avg./PTB-YM/PTB-SD) is lower than STAG-

	UD-AR	UD-DE	UD-ES	UD-ID	UD-RU	UD-ZH	UD-Avg.
STAG-A	79.58	76.17	83.13	80.51	78.74	82.03	80.03
STAG-B	82.98	78.47	85.43	82.94	81.10	83.80	82.45
STAG-C	83.60	79.78	86.14	84.18	83.22	83.60	83.42
STAG-D	85.04	84.89	88.00	86.48	86.20	84.27	85.81
STAG-E	87.18	82.25	88.32	87.16	85.73	85.60	86.04
STAG-F	**89.20**	83.76	89.32	87.99	85.99	**87.85**	87.35
STAG-G	88.77	**87.45**	**90.31**	**89.24**	**88.60**	86.25	**88.44**

Table 3.6: Supertagging results on the Universal Dependencies . Each number indicates accuracy. "UD-Avg." indicates the macro average accuracy for each supertag set over all the languages.

C accuracy (83.42/89.71/89.67% for UD-Avg./PTB-YM/PTB-SD). These two tag sets differ with respect to the encoded syntactic information for dependents. STAG-B encodes the left/right dependent possession information for each target word regardless of whether the dependents are adjunct or core arguments. In contrast, STAG-C encodes the dependency labels only if the dependents are core arguments. This suggests that whether the syntactic information relevant to adjunct arguments is encoded or not cause performance variation. Because STAG-A encodes both the dependent possession and core argument labels as well as the head information, it is more difficult to predict than STAG-B and STAG-C.

STAG-D is built by ablating the head directionality information from STAG-C, so that STAG-D is smaller than STAG-C, which leads to a performance boost relative to STAG-C. Similarly, STAG-E is built by ablating the dependency labels for core arguments from STAG-C and hence encodes only syntactic information relevant to heads. The performance boost relative to STAG-C is also observed. Comparing STAG-D with STAG-E, a noticeable difference in average accuracy is not observed, but the results within each language differ. For instance, in German (UD-DE), the accuracy for STAG-D is higher by over 2.5 points than STAG-E. On the contrary, in Arabic (UD-AR) and Chinese (UD-ZH), the accuracy for STAG-E is higher by around 1-2 points than that for STAG-D. A detailed investigation of this difference is a line of interesting future work.

	PTB-YM	PTB-SD
Baseline	92.60/91.31	92.00/89.33
Stag-A	99.06/99.01	98.47/98.27
Stag-B	**99.10/99.08**	**98.65/98.55**
Stag-C	97.77/97.77	96.88/96.78
Stag-D	98.44/98.44	96.10/96.00
Stag-E	98.45/98.43	96.88/96.76
Stag-F	98.77/97.10	**98.65**/95.23
Stag-G	97.65/97.65	96.17/96.15

Table 3.7: English dependency parsing results with gold supertags. Each number indicates UAS/LAS, in which "UAS" is the unlabeled attachment score and "LAS" is the labeled attachment score.

3.4.5 Results for Dependency Parsing with Supertags

To investigate the utility of the supertag features in transition-based dependency parsing systems, we report parsing results in various experimental settings and discuss them in detail.

Accuracy of Dependency Parsing with Gold Supertags

The utility of supertag features for dependency parsing changes according to each supertag set and supertagging accuracy. In order to check whether the proposed supertag sets and supertag feature templates capture syntactic information that is helpful for dependency parsing, we performed a parsing simulation experiment in which the condition where an arc-standard parser knows the correct (gold) supertags. In this simulated experiment, the arc-standard model receives the correct supertags and utilizes them as features. Tables 3.7 and 3.8 show the unlabeled attachment scores (UAS) and labeled attachment scores (LAS) of the baseline parsers and the supertag-integrated parsers.

In English dependency parsing (PTB-MT for Yamada and Matsumoto head rules and PTB-SD for Stanford dependencies), the unlabeled attachment scores of Stag-A/B/F reached around 99%, which indicates that the derived depen-

	UD-AR	UD-DE	UD-ES	UD-ID
Baseline	80.51/74.89	84.53/77.97	86.43/81.62	84.01/78.08
Stag-A	90.50/89.62	95.03/94.22	95.27/94.87	93.40/92.37
Stag-B	**90.95**/**90.50**	95.71/**95.05**	**95.86**/**95.65**	94.17/**93.63**
Stag-C	87.13/86.71	90.52/90.12	93.06/92.93	89.66/89.16
Stag-D	88.41/88.31	93.62/93.39	94.23/94.23	91.17/91.09
Stag-E	88.54/88.15	93.36/92.95	94.00/93.86	90.85/90.46
Stag-F	90.67/83.41	**96.78**/88.00	95.52/89.83	**94.44**/87.20
Stag-G	86.70/86.63	90.61/90.39	92.88/92.86	89.74/89.61

UD-RU	UD-ZH	UD-Avg.
83.37/77.32	83.35/79.48	83.70/78.23
92.69/91.61	96.20/95.42	93.85/93.02
93.74/**93.03**	**96.76**/**96.27**	94.53/**94.02**
89.38/88.99	95.37/94.97	90.85/90.48
91.02/90.75	95.64/95.47	92.35/92.21
90.85/90.31	95.36/94.88	92.16/91.77
94.59/86.64	95.98/91.20	**94.66**/87.71
89.57/89.32	95.41/95.41	90.82/90.70

Table 3.8: Multilingual dependency parsing results with gold supertags. Each number indicates UAS/LAS.

dency trees were almost perfect. This implies that information provided by the supertags is considerably helpful for the transition-based system to determine the times at which reduce transitions should be conducted. Consider the `RIGHT-ARC` transition, which adds an arc from the second to the top word on the stack and removes the top word from the stack. If there are any words in the buffer that depend on the word on the top of the stack, `RIGHT-ARC` should not be executed. The supertag sets Stag-A/B/F encode the head directionality and dependent possession information, which can implicitly tell the parser in which direction the second-top word in the stack has its head and whether the top word has any dependents in the buffer or not. Because this clue could supplement word

	PTB-YM	PTB-SD
Baseline	92.60/91.31	92.00/89.33
Stag-A	92.80/91.61	92.38/89.89
Stag-B	**92.94/91.67**	**92.44/89.90**
Stag-C	92.59/91.37	92.23/89.85
Stag-D	92.65/91.47	92.33/89.85
Stag-E	92.72/91.55	92.24/89.83
Stag-F	92.48/91.25	92.19/89.50
Stag-G	92.51/91.33	92.19/89.72

Table 3.9: English dependency parsing results with predicted supertags. Each number indicates UAS/LAS.

form and POS information, a parser was able to select and accumulate the correct local transition under each configuration. In fact, this result suggests that if transition-based systems knew the correct supertags that encode the head directionality and dependent possession information and could use them as features, the dependency parsing problem would be solved almost completely.

In multilingual dependency parsing, although the unlabeled attachment scores were not as high as the ones for English dependency parsing, Stag-A/B/F consistently occupied the top-3 highest UAS rankings over the six languages. The score difference between PTB and UD is likely to be caused by the data size difference, i.e., the data size of PTB is much larger than that of UD, so an investigation of the effect of increasing data size is our interesting future work. In the labeled attachment scores, Stag-F is inferior to Stag-A/B, which is consistent with English dependency parsing. This could be caused by the fact that Stag-F does not encode the head dependency label.

Accuracy of Dependency Parsing with Predicted Supertags

To investigate the utility of our supertag sets in dependency parsing in realistic situations, we performed experiments in which transition-based systems exploited automatically predicted supertags as features. Tables 3.9 and 3.9 show the unlabeled attachment scores (UAS) and labeled attachment scores (LAS) of the

	UD-AR	UD-DE	UD-ES	UD-ID
Baseline	80.51/74.89	84.53/77.97	86.43/81.62	84.01/78.08
Stag-A	81.26/76.18	85.05/79.09	87.65/83.55	85.26/79.55
Stag-B	**81.50**/**76.33**	85.01/78.96	87.72/83.48	85.43/**79.86**
Stag-C	81.17/76.22	**85.09**/79.31	**87.86**/**83.69**	84.96/79.26
Stag-D	81.13/75.94	84.86/79.09	87.65/83.48	85.13/79.57
Stag-E	81.00/76.05	85.00/**79.34**	87.50/83.43	**85.55**/79.79
Stag-F	81.05/75.51	84.76/78.03	87.00/82.19	84.80/78.93
Stag-G	81.15/76.30	85.04/79.07	**87.86**/83.57	85.09/79.47

UD-RU	UD-ZH	UD-Avg.
83.37/77.32	83.35/79.48	83.70/78.23
83.17/77.57	83.77/79.71	84.36/79.28
83.82/**78.15**	**84.33**/**80.40**	**84.64**/**79.53**
83.63/77.98	84.19/80.35	84.48/79.47
83.42/77.76	84.00/80.24	84.37/79.35
83.42/77.75	83.63/79.89	84.35/79.38
83.78/77.73	83.25/79.10	84.11/78.58
83.66/77.92	83.72/80.04	84.42/79.40

Table 3.10: Multilingual dependency parsing results with predicted supertags. Each number indicates UAS/LAS.

baseline parsers and the parsers with the supertag features.

Overall, the parsers with supertag features outperform the baseline. In particular, across the six languages of UD, a performance boost of the parsers with Stag-B is observed, yielding increases of around 1.0 point in UAS and 1.3 points in LAS. In Spanish (UD-ES) and Indonesian (UD-ID), the biggest improvements were achieved (+1.5 points in UAS and +2.0 points in LAS). In English dependency parsing (PTB-MT, PTB-SD), although the improvements of UAS/LAS were smaller than for the six languages of UD, the supertag features worked effectively. The biggest improvement (+0.34/+0.36 points in UAS/LAS of PTB-MT and +0.44/+57 points in UAS/LAS of PTB-SD) was achieved with Stag-B,

which is the same tendency as in the languages of UD.

Comparing the results with the gold and predicted supertags, the predicted supertags of STAG-F were not as effective, although the gold ones were useful for parsing. In English parsing, STAG-F was not effective for improving UAS and LAS. In the six languages of UD, although STAG-F was a little bit effective on average (+0.41 points in UAS and +35 points in LAS), the improvement was the worst of the seven supertags. In contrast, while the gold supertags of STAG-C/D/E/G did not have much predictability compared with STAG-A/F, the predicted supertags of STAG-C/D/E/G achieved almost the same UAS and LAS as STAG-A and were better than STAG-F on average. In particular, for LAS, the other supertag sets outperformed STAG-F (around +0.8 points), which suggests that it is better to encode the dependency label on the edge between the target word and its head for dependency parsing.

In addition to such head information, we wished to know which syntactic information of dependents could contribute to the improvements of UAS and LAS. To investigate this question, we compare the results of STAG-B/C/E, in which STAG-B encodes HLABEL/DIR/HASDEP, STAG-C encodes HLABLE/DIR/DLABEL, and STAG-E encodes HLABEL/DIR. Comparing STAG-C with STAG-E, they obtain much the same in UAS and LAS for both English and multilingual parsing settings. Comparing STAG-B with STAG-C, STAG-B outperformed STAG-C in both settings. These results suggest that the dependency labels of core arguments (DLABEL) are not always effective and the dependent possession information (HASDEP) contributes to the improvements of UAS and LAS.

Effects of Distance

To more deeply understand the characteristics of the parsers with supertags, we describe the parsing performance (F_1 score) according to the dependency distance, which represents the distance between a target word and its head word. Tables 3.11 and 3.12 show the F1 scores of the baseline parser and parser with STAG-B (the supertag set that achieved the highest UAS/LAS on average) for each binned dependency distance.

Overall, the supertags helped improve the identification of the longer distance dependencies. For distances over seven words (7-), a performance boost is observed, yielding an increase of around 2.0 points on average over the six languages of UD. Similarly, English dependency parsing is improved by around 1.0 point.

	PTB-YM	PTB-SD
root	95.15/96.10	93.60/94.52
1	96.72/96.87	96.34/96.53
2	94.14/94.42	93.86/94.08
3-6	90.92/91.39	90.57/91.27
7-	86.28/87.15	85.08/85.90

Table 3.11: F1 scores according to the dependency distances. Each number is "Baseline-F1/Stag-F1," "root" indicates the root identification, and "1/2/3-6/7-" indicates the distance (the number of words) between a target word and its head.

The F1 scores of the root identification (root) are improved on average as well. However, there is a performance gap between the languages. While the scores of English, German, Spanish, and Indonesian are drastically improved by the supertags, the scores of Arabic, Russian, and Chinese slightly decrease. A more detailed investigation of this is an interesting direction for future work.

3.4.6 Comparison with Existing Parsers

We compared our English parser with representative transition-based dependency parsing systems that use the Penn Treebank of Yamada & Matsumoto head rules (PTB-YM), i.e., the transition-based parser with a support vector machine of [59], pure transition-based parser of [63], dynamic-programming arc-standard parser of [26], arc-eager parser with rich non-local features of [66], transition-based joint POS tagging and parsing system of [5], and transition-based parser using neural networks of [7].

Table 3.13 shows the UAS and LAS on the test set. For our parser, we selected the highest scoring parser with Stag-B, and this parser is comparable to the previous systems. However, the transition-based systems of [5] are slightly better than our parser. One of the possible explanations is that their system is a joint model for POS tagging and dependency parsing, and hence employs higher-order features, such as third-order features, which are not utilized in our system. They

	UD-AR	UD-DE	UD-ES	UD-ID
root	93.32/93.18	86.28/87.51	85.04/89.05	87.61/89.05
1	94.27/94.59	92.44/92.84	95.85/96.09	95.09/95.45
2	80.04/81.54	87.27/88.42	91.99/93.53	84.50/85.93
3-6	74.71/76.71	84.80/85.19	85.03/85.76	79.80/82.21
7-	69.23/71.20	82.55/83.41	72.37/76.01	73.45/74.23

UD-RU	UD-ZH	UD-Avg.
90.58/90.18	79.40/79.20	87.04/88.03
93.87/94.16	94.71/94.66	94.37/94.63
87.20/87.93	86.64/87.46	86.27/87.47
80.01/79.81	83.69/84.24	81.34/82.32
67.01/67.53	75.68/77.33	72.81/74.70

Table 3.12: F1 scores according to the dependency distances. Each number is "Baseline-F1/Stag-F1," "root" indicates the root identification, and "1/2/3-6/7-" indicates the distance (the number of words) between a target word and its head.

use such features by dynamically extracting them from the partial tree structures built during the parsing process (what we call dynamic features). Although such dynamic higher-order features are available after partial tree structures are constructed, they capture a wider context, which could lead to the high performance. On the contrary, supertags capture second-order information because they consist of head and dependent information, and supertag features are always available regardless of such partial tree structures, which help improve the parsing performance. As an interesting issue, it remains for us to determine how these different types of features interact with or complement each other when both features are leveraged in a transition-based system.

	UAS	LAS
Yamada & Matsumoto 2003 [59]	90.3	-
Zhang & Clark 2008 [63]	91.4	-
Huang & Sagae 2010 [26]	92.1	-
Zhang & Nivre 2011 [66]	92.9	91.8
Bohnet & Nivre 2012 [5]	**93.38**	92.44
Chen & Manning 2014 [7]	91.8	89.6
this work	92.94	91.67

Table 3.13: UAS/LAS of dependency parsers in previous work.

3.5 Related Work

Supertags, which are lexical templates, encode linguistically rich information that imposes complex constraints in a local context [2]. While supertags have been used in frameworks based on lexicalized grammars, e.g., Lexicalized Tree-Adjoining Grammar (LTAG), Head-driven Phrase Structure Grammar (HPSG), and Combinatory Categorial Grammar (CCG), they have scarcely been utilized for dependency parsing so far. As exceptions, Foth et al. [14] and Ambati et al. [1] have used supertags for dependency parsing.

Foth et al. [14] designed supertags based on dependency structure information such as dependency labels and dependents with different levels of granularity. They automatically assigned a single supertag to each word, and the accuracy of automatically assigning their designed supertag set is 67%-84% accurate: the coarsest supertag set (35 tags) is 84.1% and the finest one (12,947 tags) is 67.6%. They then utilized the predicted supertags for dependency parsing and demonstrated that supertags improve German dependency parsing under a Weighted Constraint Dependency Grammar (WCDG), which is not data-driven parsing. In particular, the finest supertag set achieved the biggest improvement in parsing performance (+2.1 points). While they design supertags for WCDG parsing, we explore effective supertag design for data-driven and transition-based dependency parsers.

Ambati et al. [1] utilized supertags of the Combinatory Categorial Grammar

(CCG) as features for Hindi and English dependency parsers. They reported an improvement of around 0.4 points in UAS using supetag features, and argued that CCG supertags can especially improve long distance dependencies, e.g., coordination and relative clause dependencies. In contrast to their work, we develop a supertag set based on dependency structures because we believe that a supertag design based on dependency structures is more suitable for dependency parsing, rather than one based on another lexicalized grammar formalism.

3.6 Summary

In this work, we presented a supertag design framework that is flexible so that various supertag sets may be designed. Based on the framework, we instantiated various granularity supertag sets that encode rich syntactic information. In previous work, syntactic information, such as the head and dependents of a word, cannot be used as features before partial tree structures are constructed [66]. However, by exploiting the supertags as features, we can utilize fine-grained syntactic information without waiting for partial trees to be built.

To investigate the utility of these supertag features, we have performed the experiments in multilingual dependency parsing as well as English parsing. The experimental results suggest the following:

- Overall, our proposed supertag sets are effective for English and multilingual dependency parsing.
- In particular, the supertag set that encodes the *head directionality/head labels/dependent possession* achieves the highest UAS and LAS.
- Supertags contribute to the resolution of long distance dependencies.

Based on our proposed supertag design framework, we instantiated the seven supertag sets and used them as features for dependency parsers. For six languages that belong to different language branches as well as English, the supertag sets contributed to the improvements of UAS and LAS.

Comparing the results of the supertag sets, we found that in order to improve dependency parsing, it is critical to encode the head directionality, head label, and dependent possession information as supertags. In particular, the head label

information is crucial for improving LAS. In contrast, the obligatory dependent labels do not improve the results.

Analyzing the results from the aspect of dependency distances, supertags especially contributed to the improvements in long distance dependency prediction. Long distance dependencies have been regarded as a troublesome problems in dependency parsing. Our experimental results suggest that supertags could be a solution to this problem.

As our future research, we would like to investigate the interaction of supertag features with higher-order features and explore linguistic entities that capture structurally richer information, such as subtree structures.

Chapter 4

Semantic Dependency Parsing: Multi-Predicate Modeling

4.1 Introduction

Semantic dependency parsing, semantic role labeling (SRL), and predicate argument structure (PAS) analysis are a semantic analysis problem of recovering the predicate argument structure of a sentence, such as *who did what to whom*.[1]

4.1.1 Background

Figure 4.1 shows English and Japanese sentences annotated with syntactic and semantic dependencies. The two sentences have the different surface forms and syntactic dependencies. Despite the difference, they have the same predicate-argument structure: the word "John" ("ジョン") plays a role of *agent*, and the word "window" ("窓") plays a role of *theme*. This is an interesting property of predicate-argument structure.

Recently, PAS analysis in multilingual settings have attracted a considerable attention [19]. However, it is difficult to develop one unified method for multilingual PAS analysis because each language has its own unique characteristics. In fact, many of the top systems in the CoNLL-2009 shared task [19] took such

[1]These three terms "semantic dependency parsing", "semantic role labeling" and "predicate argument structure analysis" are used to indicate a similar semantic analysis task. In this thesis, since we tackle the task in Japanese, we follow the previous researches and use the term "predicate argument structure (PAS) analysis."

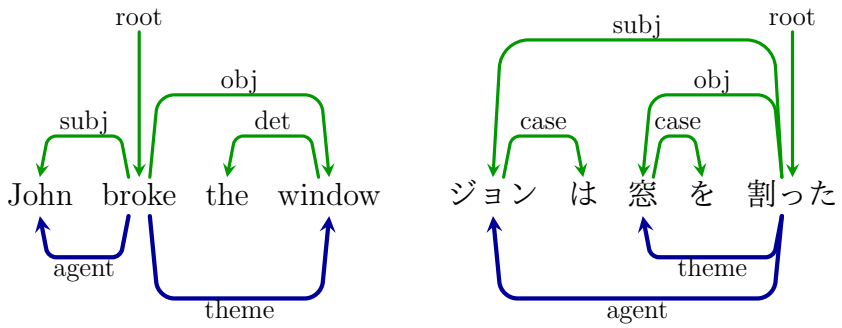


Figure 4.1: Example of two sentences that have the same predicate argument structure. Although the predicate argument structures (shown in the below part) are identical, the surface and syntactic realizations (shown in the above part) are different between the two sentences.

characteristics into account and adopted several different techniques for each language. Thus, this thesis also aims to improve PAS analysis methods for certain languages by focusing on some language phenomenon.

4.1.2 Problematic Issue: Argument Omission

In pro-drop languages such as Japanese, Chinese and Italian, arguments are often omitted in text. Such *argument omission* is regarded as one of the most problematic issues facing predicate argument structure (PAS) analysis [29, 53, 20]. Figure 4.2 illustrates an example of PAS with the argument omission problem. The omitted argument ϕ_i, called *zero pronoun*, refers to the antecedent "男 $_i$ (man$_i$)." In PAS analysis, computational systems aim to identify such antecedent as an argument for the target predicate. Hence, in the above-mentioned example, as the nominative argument for the predicate "逃走した (escaped)", the systems have to identify "男 $_i$ (man$_i$)."

What makes it difficult to identify such omitted arguments? Before the omitted argument identification, consider a case of general argument identification. For the predicate "逮捕した (arrested)", the word "警察 (police)" is the nominative argument and "男 $_i$ (man$_i$)" is the accusative argument. It is easy to identify these arguments, since a syntactic dependency between an argument and its predicate

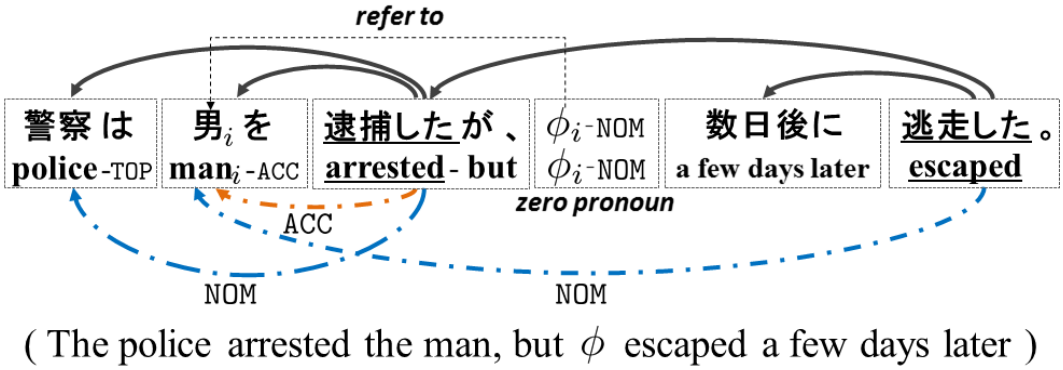


Figure 4.2: Example of Japanese predicate argument structures. The upper edges denote dependency relations, and the lower edges denote case arguments. "NOM" and "ACC" denote the nominative and accusative arguments, respectively. "ϕ_i" is a *zero pronoun*, referring to the *antecedent* "男 $_i$ (man$_i$)".

is a strong clue. By contrast, when identifying omitted arguments, such syntactic clues do not work well. For instance, when identifying the nominative argument "男 $_i$ (man$_i$)" for the predicate "逃走した (escaped)", there is no syntactic dependency between them, so that the dependency clue cannot be used as a feature. Such lack of a syntactic dependency between an argument and a predicate could be a cause of the difficulty.

4.1.3 Key Insight: Multi-Predicate Interaction

To address this issue, we aim to capture the relations between multiple predicates, called *multi-predicate interactions*. This approach is based on the linguistic intuition: the predicates in a sentence are semantically related to each other and capturing these interactions is expected to be helpful for PAS analysis.

In the example sentence in Figure 4.2, the word "男 $_i$ (man$_i$)" is the accusative argument of the predicate "逮捕した (arrested)" and is shared by the other predicate "逃走した (escaped)" as its nominative argument. Considering the semantic relation between "逮捕した (arrested)" and "逃走した (escaped)", we intuitively

know that "男 $_i$ (man$_i$)" , the person arrested by someone, is likely to be the escaper. By contrast, "警察 (police)", the person who arrested someone, is unlikely to be the escaper. That is, information about one predicate-argument relation could help to identify another predicate-argument relation.

4.1.4 Solution

To model these multi-predicate interactions, we propose the two types of PAS analysis models:

- Bipartite graph models
- Grid-type recurrent neural network models

In the following, we describe the overview of these two models.

Bipartite Graph Models
The *bipartite graph models* represent the multiple predicate-argument relations as a bipartite graph that covers all predicates and argument candidates in a sentence, and factorize the whole relation into the second-order relations. This interaction modeling results in a hard combinatorial problem because it is required to select the optimal PAS combination from all possible PAS combinations in a sentence. To solve this problem, we extend the randomized hill-climbing algorithm [65] to search all possible PAS in the space of bipartite graphs. To investigate the performance we performing experiments on the NAIST Text Corpus [28]. The experimental results show that, compared with a baseline that do not consider the multi-predicate interactions, our models achieve an improvement of 1.0-1.2 points in F1 score. Especially, they improve performance for the omitted argument identification by 2.0-2.5 points.

Grid-RNN Models
The *grid-type recurrent neural network models* take as input all predicates and argument candidates in a sentence, and automatically induce features sensitive to multi-predicate interactions. This modeling requires no complex manual feature engineering. Instead, by exploiting the feature-inducing capability of grid-type

recurrent neural networks (Grid-RNNs), the models learn effective feature representations exclusively from the word sequence information of a sentence. Experimental results on the NAIST Text Corpus demonstrate that, even without syntactic information, these models improve the baseline by 3.0-3.3 points in F1 score. Also, they outperform the bipartite graph models by about 2.0 points in F1 score. These results suggest that the proposed grid-type neural architecture effectively captures multi-predicate interactions and contributes to performance improvements.

4.1.5 Contributions

To sum up, in this chapter we make the following contributions:

- We propose (i) bipartite graph models and (ii) Grid-RNN models capturing multi-predicate interactions.
- Performing experiments on the NAIST Text Corpus [28], we demonstrate the utility of our modeling of the multi-predicate interactions for Japanese predicate argument structure analysis.

4.2 Predicate Argument Structure Analysis

4.2.1 Task Setting

Formally, given a sentence $\boldsymbol{w} = (w_1, \cdots w_T)$ and target predicates $\boldsymbol{p} = \{p_1, \cdots p_M\}$, a system predicts $\langle argument, case\ role, predicate \rangle$ tuples $\{\langle a, c, p \rangle_i\}_{i=1}^{I}$. Here, w is a word ID of the vocabulary $\mathcal{V}$, i.e. $w \in \mathcal{V}$, and p is a position index within a sentence, i.e. $p \in [1, T]$. Also, a is a position index within a sentence, i.e. $a \in [1, T]$, and c is a case role ID, i.e. $c \in \mathcal{C}$.

In the training phase, given training data $\mathcal{D}^{train} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{N^{train}}$, the target function $f : \mathcal{X} \rightarrow \mathcal{Y}$ is to be learned, where $\boldsymbol{x} = \langle \boldsymbol{w}, \boldsymbol{p} \rangle$, and $\boldsymbol{y} = \{\langle a, c, p \rangle_i\}_{i=1}^{I}$. In the evaluation phase, given evaluation data $\mathcal{D}^{eval} = \{\boldsymbol{x}_j\}_{j=1}^{N^{eval}}$, a learned function f is used to predict the tuples $\hat{\boldsymbol{y}}$.

4.2.2 Target Case Roles and Argument Types

In Japanese PAS analysis, systems aim to identify arguments with a case role for the target predicate.

Target Case Roles
The case roles to be identified are as follows:

- *Nominative* case, denoted as (`NOM`)
- *Accusative* case, denoted as (`ACC`)
- *Dative* case, denoted as (`DAT`)

Note that predicates do not always have these three case arguments. If the target predicate have no argument with a certain case role, systems output a special token `NULL`.

Target Argument Types
Arguments can be divided into the following three types according to the positions relative to their predicates [21]:

- Dep: Arguments that have direct syntactic dependency on the predicate.
- Zero: Arguments referred to by zero pronouns within the same sentence that have no direct syntactic dependency on the predicate.
- InterZero: Arguments referred to by zero pronouns outside of the same sentence.

Table 4.1 shows examples of each argument type. The first example describes a direct-dependency argument (Dep). The nominative argument "私 (I)" for the predicate "ひく (catch)" is regarded as a Dep argument, because the argument has a direct syntactic dependency on the predicate. The second example describes a zero argument (Zero). The nominative argument "男 $_i$ (man$_i$)" for the predicate "逃走する (escape)" is regarded as a Zero argument, because the argument is the antecedent of the zero pronoun and has no direct syntactic dependency on the predicate. The third example describes a inter-sentential zero argument (InterZero). The nominative argument "彼女 $_i$ (She$_i$)" for the predicate "飲

DEP	ja	私は風邪をひいた。
	en	I caught a cold.
	Prd	ひく (catch)
	Arg	NOM:私 (I), ACC:風邪 (cold), DAT:NULL
ZERO	ja	警察は男 $_i$ を逮捕したが、 (ϕ_i-NOM) 数日後に逃走した。
	en	The police arrested the man$_i$, but ϕ_i escaped a few days later.
	prd	逃走する (escape)
	arg	NOM:男 $_i$(man_i), ACC:NULL, DAT:NULL
INTERZERO	ja	彼女 $_i$ はパンを食べた。 (ϕ_i-NOM) 牛乳も飲んだ。
	en	She$_i$ ate bread. And (ϕ_i) also drank milk.
	prd	飲む (drink)
	arg	NOM=彼女 $_i$(She$_i$), ACC=牛乳 (milk), DAT=NULL

Table 4.1: Examples of each argument type. ϕ_i and word$_i$ denote the zero pronoun and its antecedent, respectively.

む (drink)" is regarded as a INTERZERO argument, because the antecedent, the nominative argument, appears outside the sentence the zero pronoun appears in.

Among these argument types, we aim to identify the DEP and ZERO arguments. In order to identify inter-sentential arguments (INTERZERO), a much broader space must be searched (e.g., the whole document), resulting in a much more complicated analysis than intra-sentential arguments.[2] Owing to this complication, this thesis focuses exclusively on intra-sentential argument analysis, i.e. DEP and ZERO.

[2]The F1 score of inter-sentential argument analysis remains 10-20% [57, 32, 53].

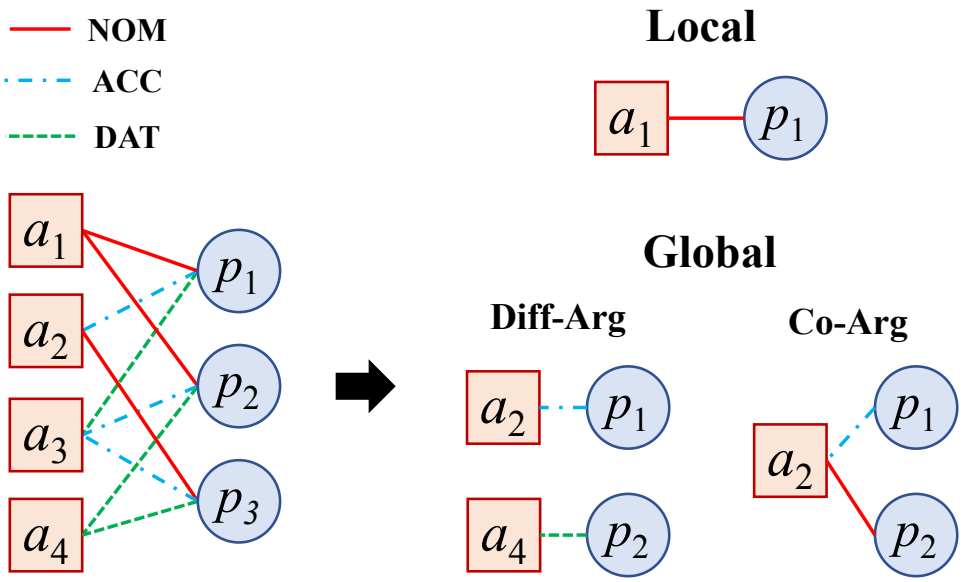


Figure 4.3: Intuitive image of a *predicate-argument graph*. This graph is factorized into the local and global features. The different line color/style indicate different cases.

4.3 Bipartite Graph Models

4.3.1 A Predicate-Argument Graph

We define predicate argument relations by exploiting a bipartite graph. Figure 4.3 illustrates an example of the graph, called PREDICATE-ARGUMENT GRAPH (PA graph). The nodes of the graph consist of two disjoint sets: the left one is a set of *argument candidates* and the right one is a set of *predicates*. Each predicate node has three distinct edges corresponding to nominative (NOM), accusative (ACC), and dative (DAT) cases. Each edge with a case role label joins a argument candidate node with a predicate node, which represents a case argument of a predicate. For instance, in Figure 4.3 a_1 is the nominative argument of the predicate p_1, and a_3 is the accusative argument of the predicate p_2.

Formally, a PA graph consists of three elements: the node set consisting of

argument candidates A, the node set consisting of predicates P, and the set of edges E:

$$\text{PA graph: } \langle A, P, E \rangle$$

Each element is defined as follows:

$$\begin{aligned} A &= \{a_1, \cdots, a_n, a_{n+1} = \texttt{NULL}\} \\ P &= \{p_1, \cdots, p_m\} \\ E &= \{\langle a, p, c \rangle \mid deg(p, c) = 1, \\ &\qquad \forall a \in A,\ \forall p \in P,\ \forall c \in C\ \} \end{aligned}$$

In the graph, the left nodes correspond to A and the right ones correspond to P. A consists of n argument candidates, $\{a_1, \cdots, a_n\}$, and a dummy node a_{n+1}. This dummy node a_{n+1} is defined for the cases where the predicate requires no case argument or the required case argument does not appear in the sentence. P consists of m predicates, $\{p_1, \cdots, p_m\}$. E is the set of edges connected between A and P. An edge $e \in E$ is represented by a tuple $\langle a, p, c \rangle$, indicating the edge with a case role c joining a argument candidate node a and a predicate node p. An admissible PA graph satisfies the constraint $deg(p, c) = 1$, representing that each predicate node p has only one edge with a case role c.

To identify the whole predicate argument structures (PAS) for a sentence x, we predict the PA graph with an edge set corresponding to the correct PAS from the admissible PA graph set $G(x)$ based on a score associated with a PA graph y as follows:

$$\tilde{y} = \underset{y \in G(x)}{\text{argmax}}\ Score(x, y)$$

A scoring function $Score(x, y)$ receives a sentence x and a candidate graph y as its input, and returns a scalar value.

In this thesis, we propose the two scoring functions as analysis models based on different assumptions:

- *Per-case joint model*: Assumes the interaction between multiple predicates (*predicate interaction*) and the independence between case roles.

- *All-cases joint model*: Assumes the interaction between case roles (*case interaction*) as well as the *predicate interaction*.

In the following subsections, we describe each of these models in more detail.

4.3.2 Per-Case Joint Model

The *per-case joint model* assumes that different case roles are independent from each other. However, for each case, the interactions between multiple predicates are considered jointly.

We define the score of a PA graph y to be the sum of the scores for each case role c of the set of the case roles C:

$$Score_{per}(x, y) = \sum_{c \in C} Score_c(x, y) \tag{4.1}$$

Scores for each case role are defined as the dot products between a weight vector $\boldsymbol{\theta_c}$ and a feature vector $\boldsymbol{\phi_c}(x, E(y, c))$:

$$Score_c(x, y) = \boldsymbol{\theta_c} \cdot \boldsymbol{\phi_c}(x, E(y, c)) \tag{4.2}$$

where $E(y, c)$ is the edge set associated with a case role c in the candidate graph y, and the feature vector is defined on the edge set.

The edge set $E(y, c)$ in Equation 4.2 is utilized for the two types of features: **local features** and **global features**. These features are inspired by [24] and defined as follows:

$$\boldsymbol{\theta_c} \cdot \boldsymbol{\phi_c}(x, E(y, c)) = \sum_{e \in E(y,c)} \boldsymbol{\theta_c} \cdot \boldsymbol{\phi_\ell}(x, e) + \boldsymbol{\theta_c} \cdot \boldsymbol{\phi_g}(x, E(y, c)) \tag{4.3}$$

where $\boldsymbol{\phi_\ell}(x, e)$ denotes a local feature vector, and $\boldsymbol{\phi_g}(x, E(y, c))$ a global feature vector.

The local feature vector $\boldsymbol{\phi_\ell}(x, e)$ is defined on each edge e in the edge set $E(y, c)$ and a sentence x, which captures a predicate-argument pair. Consider the case where the per-case model estimates the nominative arguments in Figure 4.3. To compute the score of the edge set with the nominative case $c = \texttt{NOM}$, the model uses the following edges:

$$\{\, e_{a_1 p_1},\ e_{a_1 p_2},\ e_{a_2 p_3} \,\}$$

Each of these edges is taken as input by $\boldsymbol{\phi_\ell}(x, e)$ in Equation 4.3, and the resulting feature vector is multiplied with the weight vector $\boldsymbol{\theta}_{c=\texttt{NOM}}$.

The global feature vector $\boldsymbol{\phi_g}(x, E(y, c))$ is defined on the edge set $E(y, c)$, and enables the model to utilize linguistically richer information over multiple predicate-argument pairs. In this thesis, we exploit *second-order* relations, similar to the second-order edge factorization of dependency trees [42]. We make a set of edge pairs E_{pair} by combining two edges e_i, e_j in the edge set $E(y, c)$, as follows:

$$E_{pair} = \{\ \{e_i, e_j\} \mid \forall e_i, e_j \in E(y, c),\ e_i \neq e_j\ \}$$

For instance, in the PA graph in Figure 4.3, to compute the score of the nominative arguments $c = \texttt{NOM}$, we make three edge pairs:

$$E_{pair} = \{\ \{e_{a_1p_1},\ e_{a_1p_2}\}, \{e_{a_1p_1},\ e_{a_2p_3}\}, \{e_{a_1p_2},\ e_{a_2p_3}\}\ \}$$

This set of the edge pairs is taken as input by $\boldsymbol{\phi_g}(x, E_{pair})$ in Equation 4.3, and the resulting feature vector is multiplied with the weight vector $\boldsymbol{\theta}_{c=\texttt{NOM}}$. In the same way, for the accusative and dative cases, their scores are computed. Then, we obtain the resulting score of the PA graph by summing up the scores of the local and global features. If we do not consider the global features, the model reduces to a per-case local model similar to the pointwise model [32].

4.3.3 All-Cases Joint Model

While per-case joint model assumes the *predicate interaction* with the independence between case roles, *all-cases joint model* assumes the *case interaction* together with the *predicate interaction*. Our graph-based formulation is very flexible and easily enables the extension of per-case joint model to all-cases joint model. Therefore, we extend per-case joint model to all-cases joint model to capture the interactions between predicates and all case arguments in a sentence.

We define the score of a PA graph y based on the local and global features as follows:

$$Score_{all}(x, y) = \sum_{e \in E(y)} \boldsymbol{\theta} \cdot \boldsymbol{\phi_\ell}(x, e) + \boldsymbol{\theta} \cdot \boldsymbol{\phi_g}(x, E(y)) \tag{4.4}$$

where $E(y)$ is the edge set associated with all the case roles on the candidate graph y, $\boldsymbol{\phi_\ell}(x, e)$ is the local feature vector defined on each edge e in the edge set $E(y)$, and $\boldsymbol{\phi_g}(x, E(y))$ is the global feature vector defined on the edge set $E(y)$.

Consider the PA graph in Figure 4.3. The local features are extracted from each of the following edges:

$$\texttt{NOM} : \{e_{a_1p_1}, e_{a_1p_2}, e_{a_2p_3}\}$$

$$\texttt{ACC} : \{e_{a_2p_1}, e_{a_3p_2}, e_{a_3p_3}\}$$

$$\texttt{DAT} : \{e_{a_3p_1}, e_{a_4p_2}, e_{a_4p_3}\}$$

For the global features, we make a set of edge pairs E_{pair} by combining two edges (e_i, e_j) in the edge set $E(y)$, like per-case joint model. However, in the all-cases joint model, the global features may involve different cases, i.e., mixing edges with different case roles. For instance, for the PA graph in Figure 4.3, we make the edge pairs $\{e_{a_1p_1},\ e_{a_2p_1}\}$, $\{e_{a_3p_1},\ e_{a_1p_2}\}$, $\{e_{a_3p_2},\ e_{a_4p_3}\}$, and so on. From these edge pairs, we extract information as global features to compute a graph score.

4.3.4 Features

Features are extracted based on *feature templates*, which are functions that draw information from the given entity. Look at the following feature template example:

$$\phi_{100} = a.ax \circ p.vo$$

This template returns a conjunction of two atomic features $a.ax$ and $p.vo$, where $a.ax$ is an auxiliary word attached to a argument candidate and $p.vo$ is the voice of a predicate. Using such templates, we draw feature information from the given entity.

To characterize a PA graph, we draw some linguistic information associated with the edge by using feature templates. In this thesis, to draw the global features, we design global feature templates based on the following substructures:

- **Diff-Arg**: Two predicate and argument nodes.
- **Co-Arg**: Two predicate nodes and one common argument node.

These substructures are depicted in the right part of Figure 4.3. The Diff-Arg represents that the two predicates have different argument candidates. The Co-Arg represents that the two predicates share the same argument candidate. In the following, we describe the features based on each substructure in more detail.

Diff-Arg Features
The feature templates based on the Diff-Arg structure are three types: **PAIR** (a pair of predicate-argument relation), **TRIANGLE** (a predicate and its two arguments relation), and **QUAD** (two predicate-argument relations).

- **PAIR** These feature templates denote where the target argument is located relative to another argument and the two predicates in the Diff-Arg structure. We combine the relative position information with the auxiliary words and the voice of the two predicates.
- **TRIANGLE** These feature templates capture the interactions between three elements: two arguments and a predicate. Like the PAIR feature templates, we encode the relative position information of two arguments and a predicate with the auxiliary words and voice.
- **QUAD** When we judge if a candidate argument takes part in a case role of a predicate, it would be beneficial to grasp information of another predicate-argument pair. The QUAD feature templates capture the mutual relation between four elements: two arguments and predicates. We encode the relative position information, the auxiliary words, and the voice.

Co-Arg Features
To identify predicates that take ZERO arguments, we set two feature types, **BI-PREDS** and **DEP-REL**, based on the Co-Arg structure.

- **BI-PREDS** For identifying an implicit argument of a predicate, information of another semantically-related predicate in the sentence could be effective. We utilize bi-grams of the regular forms of the two predicates in

the Co-Arg structure to capture the predicates that are likely to share the same argument in the sentence.

- **DEP-REL** We set five distinct feature templates to capture dependency relations between the shared argument and the two predicates. If two elements have a direct dependency relation, we encode its dependency relation with the auxiliary words and the voice.

Formally, all these global feature templates are defined in Table B.1. The proposed models utilize these global features as well as the local features[3].

4.3.5 Inference and Training

Inference

Global features make the inference of finding the maximum scoring PA graph more difficult. For searching the graph with the highest score, we propose two greedy search algorithms by extending the *randomized hill-climbing* algorithm proposed in [65], which has been shown to achieve the state-of-the-art performance in dependency parsing.

Figure 4.4 describes the pseudo code of our proposed algorithm for per-case joint model. Firstly, we set an initial PA graph $y^{(0)}$ sampled uniformly from the set of admissible PA graphs $G(x)$ (line 1 in Figure 4.4). Then, the union Y_c is constructed from the set of neighboring graphs with a case $NeighborG(y^{(t)}, c)$, which is a set of admissible graphs obtained by changing one edge with the case c in $y^{(t)}$, and the current graph $y^{(t)}$ (line 5). The current graph $y^{(t)}$ is updated to a higher scoring graph $y^{(t+1)}$ selected from the union Y_c (line 6). The algorithm continues until no more score improvement is possible by changing an edge with the case c in $y^{(t)}$ (line 8). This repetition is executed for other case roles in the same manner. As a result, we can get a locally optimal graph $\tilde{y}$.

Figure 4.5 describes the pseudo code of the algorithm for all-cases joint model. The large part of the algorithm is the same as that for per-case joint model. The difference is that the union Y consists of the current graph $y^{(t)}$ and the

[3]As the local features, we use the ones used in a model of the previous work [21]

Input: the set of cases to be analyzed C,
parameter $\boldsymbol{\theta_c}$,
sentence x
Output: a locally optimal PA graph $\tilde{y}$

1: Sample a PA graph $y^{(0)}$ from $G(x)$
2: $t \leftarrow 0$
3: **for** each case $c \in C$ **do**
4: **repeat**
5: $Y_c \leftarrow NeighborG(y^{(t)}, c) \cup y^{(t)}$
6: $y^{(t+1)} \leftarrow \underset{y \in Y_c}{\operatorname{argmax}} \boldsymbol{\theta_c} \cdot \boldsymbol{\phi_c}(x, E(y, c))$
7: $t \leftarrow t + 1$
8: **until** $y^{(t)} = y^{(t+1)}$
9: **end for**
10: **return** $\tilde{y} \leftarrow y^{(t)}$

Figure 4.4: Randomized hill-climbing for per-case joint model.

neighboring graph set obtained by changing one edge in $y^{(t)}$ regardless of case roles (line 4 in Figure 4.5), and that the iteration process for each case role (line 3 in Figure 4.4) is removed. The algorithm also continues until no more score improvement is possible by changing an edge in $y^{(t)}$, resulting in a locally optimal graph $\tilde{y}$.

Following Zhang et al. (2014), for a given sentence x, we repeatedly run these algorithms with K consecutive restarts. Each run starts with initial graphs randomly sampled from the set of admissible PA graphs $G(x)$, so that we obtain K local optimal graphs by K restarts. Then the highest scoring one of K graphs is selected for the sentence x as the result. Each run of the algorithms is independent from each other, so that multiple runs are easily executable in parallel.

Training

Given a training data set $D = \{(\hat{x}_i, \hat{y}_i)\}_{i=1}^{N}$, the weight vectors $\boldsymbol{\theta}$ ($\boldsymbol{\theta_c}$) in the scoring functions of the joint models are estimated by using machine learning

Input: the set of cases to be analyzed C,
parameter $\boldsymbol{\theta}$,
sentence x

Output: a locally optimal PA graph $\tilde{y}$

Sample a PA graph $y^{(0)}$ from $G(x)$
$t \leftarrow 0$
repeat
  $Y \leftarrow NeighborG(y^{(t)}) \cup y^{(t)}$
  $y^{(t+1)} \leftarrow \underset{y \in Y}{\text{argmax}}\ \boldsymbol{\theta} \cdot \boldsymbol{\phi}(x, E(y))$
  $t \leftarrow t + 1$
until $y^{(t)} = y^{(t+1)}$
return $\tilde{y} \leftarrow y^{(t)}$

Figure 4.5: Randomized hill-climbing for all-cases joint model.

techniques. We adopt averaged perceptron [9] with a max-margin technique:

$$\forall i \in \{1, ..., N\},\ y \in G(x_i),$$
$$Score(\hat{x}_i, \hat{y}_i) \geq Score(\hat{x}_i, y) + \|\hat{y}_i - y\|_1 - \xi_i$$

where $\xi_i \geq 0$ is the slack variable and $\|\hat{y}_i - y\|_1$ is the Hamming distance between the gold PA graph $\hat{y}_i$ and a candidate PA graph y of the admissible PA graphs $G(x_i)$. Following Zhang et al. (2014), we select the highest scoring graph $\tilde{y}$ as follows:

$$\text{TRAIN} : \tilde{y} = \underset{y \in G(\hat{x}_i)}{\text{argmax}} \{Score(\hat{x}_i, y) + \|\hat{y}_i - y\|_1\}$$
$$\text{TEST} : \tilde{y} = \underset{y \in G(x)}{\text{argmax}} \{Score(x, y)\}$$

Using the weight vector tuned by the training, we perform analysis on a sentence x in the test set.

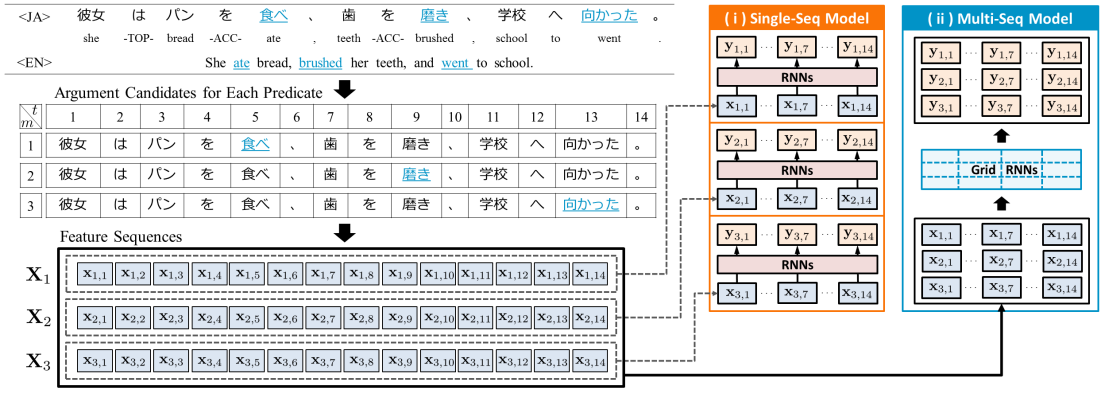


Figure 4.6: Overview of neural models: (i) *single-sequence* and (ii) *multi-sequence* models.

4.4 Grid RNN Models

We propose two neural models: (i) *single-sequence* and (ii) *multi-sequence* models, which automatically induce features sensitive to multi-predicate interactions exclusively from the word sequence information of a sentence. Figure 4.6 illustrates the overview of these models. The single-sequence model takes as input a sequence of features for one predicate, and captures the interactions between argument candidates and the predicate using recurrent neural networks (RNNs). By contrast, the multi-sequence model takes as input all sequences of features for all the predicates in a sentence at a time, and captures the interactions between argument candidates and the predicates using grid-type recurrent neural networks (Grid-RNNs). In the following subsections, we describe these two models in more detail.

4.4.1 Single-Sequence Model

The single-sequence model exploits stacked bidirectional RNNs (Bi-RNN) [54, 16, 17, 67]. Figure 4.7 shows the overall architecture, which consists of the following three components:

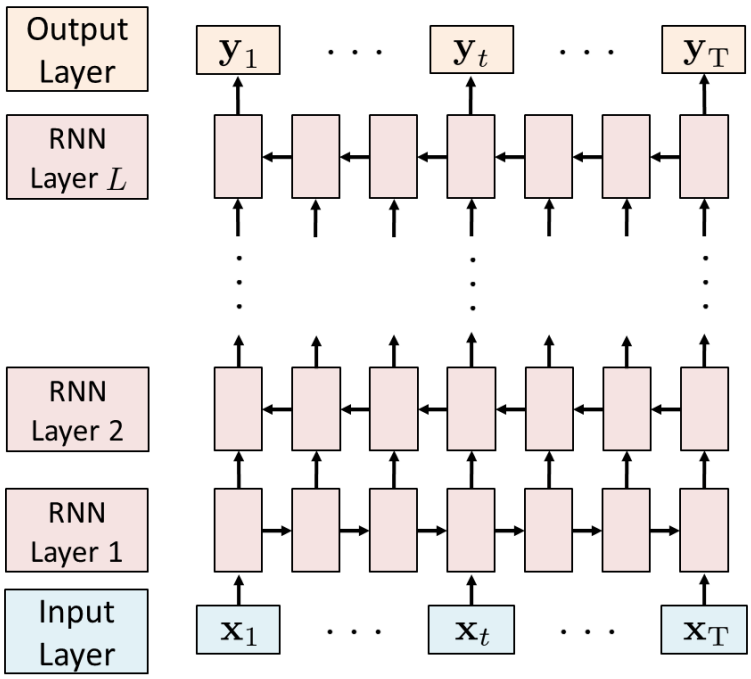


Figure 4.7: Overall architecture of the single-sequence model. This model consists of three components: (i) Input Layer, (ii) RNN Layer and (iii) Output Layer.

- **Input Layer**: Map each word to a feature vector representation.
- **RNN Layer**: Produce high-level feature vectors using Bi-RNNs.
- **Output Layer**: Compute the probability of each case label for each word using the softmax function.

In the following, we describe each of these three components in detail.

Input Layer

Given an input sentence $w_{1:\mathrm{T}} = (w_1, \cdots, w_\mathrm{T})$ and a predicate p, each word w_t is mapped to a feature representation $\mathbf{x}_t$, which is the concatenation ($\oplus$) of three

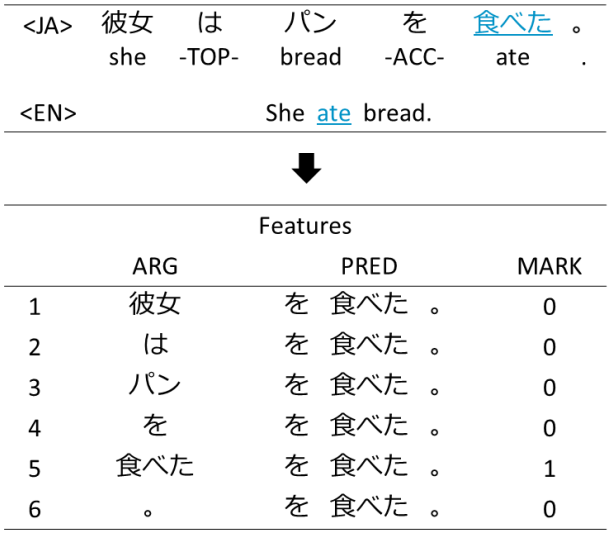


Figure 4.8: Example of feature extraction. The underlined word is the target predicate. From the sentence "彼女はパンを食べた。(She ate bread.)", three types of features are extracted for the target predicate "食べた (ate)".

types of vectors:

$$\mathbf{x}_t = \mathbf{x}_t^{arg} \oplus \mathbf{x}_t^{pred} \oplus \mathbf{x}_t^{mark} \tag{4.5}$$

where each vector is based on the following atomic features inspired by [67]:

`ARG:` Word index of each word.

`PRED:` Word index of the target predicate and the words around the predicate.

`MARK:` Binary index that represents whether or not the word is the predicate.

Figure 4.8 presents an example of the atomic features. For the `ARG` feature, we extract a word index $x^{word} \in \mathcal{V}$ for each word. Similarly, for the `PRED` feature, we extract each word index x^{word} for the C words taking the target predicate at the center, where C denotes the window size. The `MARK` feature $x^{mark} \in \{0, 1\}$ is a binary value that represents whether or not the word is the predicate.

Then, using feature indices, we extract feature vector representations from each embedding matrix. Figure 4.9 shows the process of creating the feature

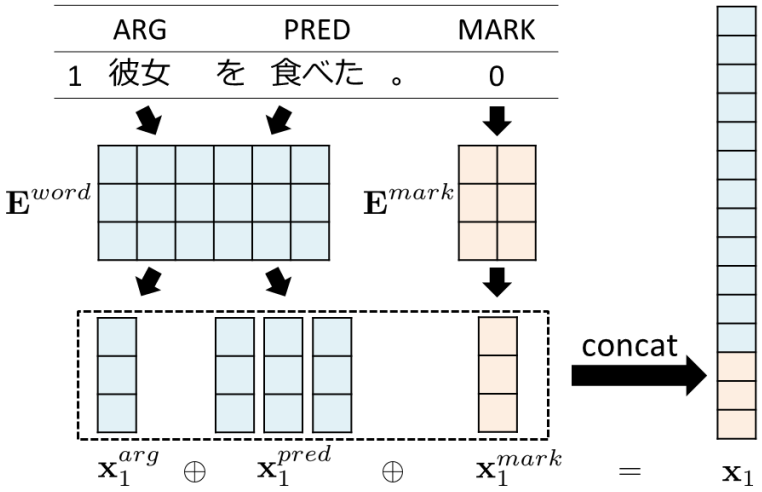


Figure 4.9: Example of the process of creating a feature vector. The extracted features are mapped to each vector, and all the vectors are concatenated into one feature vector.

vector $\mathbf{x}_1$ for the word w_1 "彼女 (she)". We set two embedding matrices: (i) a word embedding matrix $\mathbf{E}^{word} \in \mathbb{R}^{d_{word} \times |\mathcal{V}|}$, and (ii) a mark embedding matrix $\mathbf{E}^{mark} \in \mathbb{R}^{d_{mark} \times 2}$. From each embedding matrix, we extract the corresponding column vectors and concatenate them as a feature vector $\mathbf{x}_t$ based on Eq. 4.5.

Each feature vector $\mathbf{x}_t$ is multiplied with a parameter matrix $\mathbf{W}_x$:

$$\mathbf{h}_t^{(0)} = \mathbf{W}_x \, \mathbf{x}_t \tag{4.6}$$

The vector $\mathbf{h}_t^{(0)}$ is given to the first RNN layer as input.

RNN Layer

In the RNN layers, feature vectors are updated recurrently using Bi-RNNs. Bi-RNNs process an input sequence in a left-to-right manner for odd-numbered layers and in a right-to-left manner for even-numbered layers. By stacking these layers, we can construct the deeper network structures.

Stacked Bi-RNNs consist of L layers, and the hidden state in the layer $\ell \in (1, \cdots, L)$ is calculated as follows:

$$\mathbf{h}_t^{(\ell)} = \begin{cases} g^{(\ell)}(\mathbf{h}_t^{(\ell-1)},\ \mathbf{h}_{t-1}^{(\ell)}) & (\ell = \text{odd}) \\ g^{(\ell)}(\mathbf{h}_t^{(\ell-1)},\ \mathbf{h}_{t+1}^{(\ell)}) & (\ell = \text{even}) \end{cases} \tag{4.7}$$

Both of the odd- and even-numbered layers receive $\mathbf{h}_t^{(\ell-1)}$, the t-th hidden state of the $\ell - 1$ layer, as the first input of the function $g^{(\ell)}$, which is an arbitrary function [4]. For the second input of $g^{(\ell)}$, odd-numbered layers receive $\mathbf{h}_{t-1}^{(\ell)}$, whereas even-numbered layers receive $\mathbf{h}_{t+1}^{(\ell)}$. By calculating the hidden states until the L-th layer, we obtain a hidden state sequence $\mathbf{h}_{1:\mathrm{T}}^{(L)} = (\mathbf{h}_1^{(L)}, \cdots, \mathbf{h}_{\mathrm{T}}^{(L)})$. Using each vector $\mathbf{h}_t^{(L)}$, we calculate the probability of case labels for each word in the output layer.

Output Layer

For the output layer, multi-class classification is performed using the softmax function:

$$\mathbf{y}_t = \mathrm{softmax}(\mathbf{W}_y\, \mathbf{h}_t^{(L)})$$

where $\mathbf{h}_t^{(L)}$ denotes a vector representation propagated from the last RNN layer (Fig 4.7). Each element of $\mathbf{y}_t$ is a probability value corresponding to each label. The label with the maximum probability among them is output as a result. In this work, we set five labels: `NOM`, `ACC`, `DAT`, `PRED`, `null`. `PRED` is the label for the predicate, and `null` denotes a word that does not fulfill any case role.

4.4.2 Multi-Sequence Model

Whereas the single-sequence model assumes independence between predicates, the multi-sequence model assumes *multi-predicate interactions*. To capture such interactions between all predicates in a sentence, we extend the single-sequence model to the multi-sequence model using Grid-RNNs [18, 33]. Figure 4.10 presents the overall architecture for the multi-sequence model, which consists of three components:

[4] In this work, we used the Gated Recurrent Unit (GRU) [8] as the function $g^{(\ell)}$.

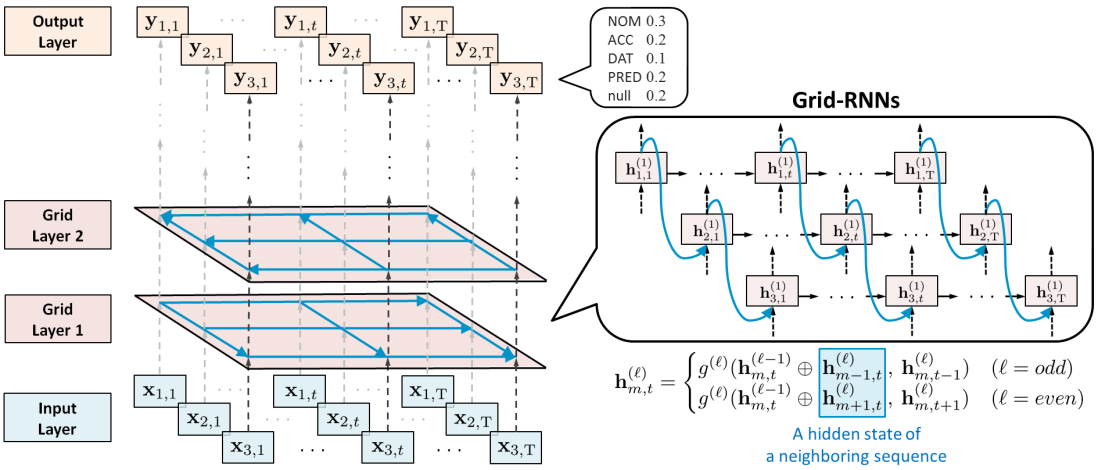


Figure 4.10: Overall architecture of the multi-sequence model: an example of three sequences.

- **Input Layer**: Map input words to M sequences of feature vectors for M predicates.
- **Grid Layer**: Update the hidden states over different sequences using Grid-RNNs.
- **Output Layer**: Compute the probability of each case label for each word using the softmax function.

In the following, we describe these three components in detail.

Input Layer

The multi-sequence model takes as input a sentence $w_{1:\mathrm{T}} = (w_1, \cdots, w_\mathrm{T})$ and all predicates $\{p_m\}_1^M$ in the sentence. For each predicate p_m, the input layer creates a sequence of feature vectors $\mathbf{X}_m = (\mathbf{x}_{m,1}, \cdots, \mathbf{x}_{m,\mathrm{T}})$ by mapping each input word w_t to a feature vector $\mathbf{x}_{m,t}$ based on Eq 4.5. That is, for M predicates, M sequences of feature vectors $\{\mathbf{X}_m\}_1^M$ are created.

Then, using Eq. 4.6, each feature vector $\mathbf{x}_{m,t}$ is mapped to $\mathbf{h}^{(0)}_{m,t}$, and a feature sequence is created for a predicate p_m, i.e., $\mathbf{H}^{(0)}_m = (\mathbf{h}^{(0)}_{m,1}, \cdots, \mathbf{h}^{(0)}_{m,\mathrm{T}})$. Consequently, for M predicates, we obtain M feature sequences $\{\mathbf{H}^{(0)}_m\}^M_1$.

Grid Layer

(a) Inter-Sequence Connections

For the grid layers, we use Grid-RNNs to propagate the feature information over the different sequences (*inter-sequence connections*). The figure on the right in Figure 4.10 shows the first grid layer. The hidden state is recurrently calculated from the upper-left ($m = 1, t = 1$) to the lower-right ($m = M, t = \mathrm{T}$).

Formally, in the ℓ-th layer, the hidden state $\mathbf{h}^{(\ell)}_{m,t}$ is calculated as follows:

$$\mathbf{h}^{(\ell)}_{m,t} = \begin{cases} g^{(\ell)}(\mathbf{h}^{(\ell-1)}_{m,t} \oplus \mathbf{h}^{(\ell)}_{m-1,t}, \mathbf{h}^{(\ell)}_{m,t-1}) & (\ell = \text{odd}) \\ g^{(\ell)}(\mathbf{h}^{(\ell-1)}_{m,t} \oplus \mathbf{h}^{(\ell)}_{m+1,t}, \mathbf{h}^{(\ell)}_{m,t+1}) & (\ell = \text{even}) \end{cases}$$

This equation is similar to Eq. 4.7. The main difference is that the hidden state of a neighboring sequence, $\mathbf{h}^{(\ell)}_{m-1,t}$ (or $\mathbf{h}^{(\ell)}_{m+1,t}$), is concatenated ($\oplus$) with the hidden state of the previous ($\ell - 1$) layer, $\mathbf{h}^{(\ell-1)}_{m,t}$, and is taken as input of the function $g^{(\ell)}$.

In the figure on the right in Figure 4.10, the blue curved lines represent the inter-sequence connections. Taking as input the hidden states of neighboring sequences, the network propagates feature information over multiple sequences (i.e., predicates). By calculating the hidden states until the L-th layer, we obtain M sequences of the hidden states, i.e., $\{\mathbf{H}^{(L)}_m\}^M_1$, in which $\mathbf{H}^{(L)}_m = (\mathbf{h}^{(L)}_{m,1}, \cdots, \mathbf{h}^{(L)}_{m,\mathrm{T}})$.

(b) Residual Connections

As more layers are stacked, it becomes more difficult to learn the model parameters, owing to various challenges such as the vanishing gradient problem [51]. In this work, we integrate residual connections [22, 58] with our networks to form connections between layers. Specifically, the input vector $\mathbf{h}^{(\ell-1)}_{m,t}$ of the ℓ-th layer is added to the output vector $\mathbf{h}^{(\ell)}_{m,t}$. Residual connections can also be applied to the single-sequence model. Thus, we can perform experiments on both models with/without residual connections.

Output Layer

As with the single-sequence model, we use the softmax function to calculate the probability of the case labels of each word w_t for each predicate p_m:

$$\mathbf{y}_{m,t} = \text{softmax}(\mathbf{W}_y \, \mathbf{h}^{(L)}_{m,t})$$

where $\mathbf{h}^{(L)}_{m,t}$ is a hidden state vector calculated in the last grid layer.

4.4.3 Training

We train the model parameters by minimizing the cross-entropy loss function:

$$\mathcal{L}(\boldsymbol{\theta}) = -\sum_n \sum_t \log P(y_t|x_t) \; + \frac{\lambda}{2}||\boldsymbol{\theta}||^2 \tag{4.8}$$

where θ is a set of model parameters, and the hyper-parameter λ is the coefficient governing the L2 weight decay.

4.5 Experiment

4.5.1 Experimental Settings

Dataset

We evaluate our proposed models on the NAIST Text Corpus 1.5, which consists of 40,000 sentences of Japanese newspaper text [28]. While previous work has adopted the version 1.4 beta, we adopt the latest version. The major difference between version 1.4 beta and 1.5 is revision of dative case (corresponding to Japanese case particle "ni"). In 1.4 beta, most of adjunct usages of "ni" are mixed up with the argument usages of "ni", making the identification of dative cases seemingly easy. Therefore, our results are not directly comparable with previous work.

We adopt standard train/dev/test splits [57] as follows:

Train: Articles: Jan 1-11, Editorials: Jan-Aug
Dev: Articles: Jan 12-13, Editorials: Sept

Test: Articles: Jan 14-17, Editorials: Oct-Dec

We exclude inter-sentential arguments (InterZero) in our experiments. The features used in our bipartite graph models make use of the annotated POS tags, phrase boundaries, and dependency relations annotated in the NAIST Text Corpus. We do not use any external resources.

Comparative Models

We investigate and compare the following models:

- Baseline: a pointwise model proposed in [32]
- PCJoint: the Per-Case Joint Model described in Section 4.3
- ACJoint: the All-Cases Joint Model described in Section 4.3
- SingleSeq: the single sequence model described in Section 4.4
- MultiSeq: the multiple sequence model described in Section 4.4

As the baseline, we adopt the pointwise model (using only local features) proposed in [32]. This model estimates the likelihood that each argument candidate plays a case role of the target predicate and independently selects the highest scoring one per predicate.

Implementation Details

For our bipartite graph models with hill-climbing, we set the number of the random restarts at 10, which almost reaches convergence [5]. For implementations of our Grid-RNN models, we used a deep learning library, Theano [3]. The parameters were optimized using the stochastic gradient descent method (SGD) via a mini-batch. Table B.2 lists the hyper-parameters. The initial values of all the parameters were sampled according to a uniform distribution from $[-\frac{\sqrt{6}}{\sqrt{row+col}}, \frac{\sqrt{6}}{\sqrt{row+col}}]$, where *row* and *col* are the number of rows and columns of each matrix, respectively. Also, words with a frequency of 2 or more in the training set were mapped to each word ID, and the remaining words were mapped to the unknown word ID.

[5] Performance did not change when increasing the number of restarts

	Dep	Zero	All
Baseline	85.06	41.65	78.15
PCJoint	85.79	43.60	78.91
ACJoint	86.07	44.09	79.23
SigleSeq	88.10	46.10	81.15
MultiSeq	**88.17**	**47.12**	**81.42**

Table 4.2: Test F1 scores on the NAIST Text Corpus 1.5. Baseline is the reimplemented model of [32], PCJoint is the Per-Case Joint Model in Section 4.3.2, ACJoint is the ALL-Cases Joint Model in Section 4.3.2, SingleSeq is the single-sequence model in Section 4.4.1, and MultiSeq is the multi-sequence model in Section 4.4.2.

4.5.2 Results

Table 4.2 presents F1 scores on the test set. All our four models outperformed the baseline. Our bipartite models improved overall F1 scores by around 1.0 point. In particular, the models yield a considerable improvement in F1 score of 2.0-2.5% for zero arguments (Zero), which have no syntactic dependency on their predicate and are regarded as one of the problematic issues in Japanese PAS analysis. Also, our neural models achieved a significant improvement of 3.0-3.3% for overall arguments and 4.5-5.5% for zero arguments. These results shows that capturing multi-predicate interactions is particularly effective for Japanese PAS analysis.

Per-Case Joint Model vs All-Cases Joint Model
Comparing the bipartite models, All-Cases Joint Model (ACJoint) outperformed Per-Case Joint Model (PCJoint) in terms of the overall F1 scores (79.23% vs 78.91%). For each argument type (Dep and Zero), All-Cases Joint Model achieved better results, i.e., 86.07% vs 85.79% for direct dependency arguments (Dep) and 44.09% vs 43.60% for zero arguments (Zero). This suggests that capturing case interactions improves performance of Japanese PAS analysis.

	Feature	Dep	Zero	All
PCJoint	*local*	84.59	42.55	77.89
	+ *global*	85.51	44.54	78.85
ACJoint	*local*	84.17	41.33	77.43
	+ *global*	85.92	44.45	79.17

Table 4.3: Global vs local features on the development set in F1 score. PCJoint and ACJoint denotes the Per-Case and All-Cases Joint Model, respectively.

Single-Sequence Model vs Multi-Sequence Model

Comparing the neural models, the multi-sequence model (MultiSeq) outperformed the single-sequence model (SingleSeq) in terms of the overall F1 scores (81.42% vs 81.15%). While for direct dependency arguments (Dep), the multi-sequence model achieved slightly better results (88.10% vs 88.17%), for zero arguments (Zero), the multi-sequence model yields around 1.0% improvement (46.10% vs 47.12%). These results demonstrate that the grid-type neural architecture can effectively capture multi-predicate interactions by connecting the sequences of the argument candidates for all predicates in a sentence.

Bipartite Graph Models vs Neural Models

Comparing between the bipartite graph and neural models, the neural models outperformed the bipartite models in F1 score overall, i.e., 81.15% and 81.42% vs 78.91% and 79.23%. In particular, for zero arguments (*Zero*), compared with All-Cases Joint Model (ACJoint), the single-sequence model achieved around 2.0% improvement (46.10% vs 44.09%), and the multi-sequence model yielded around 3.0% (47.12% vs 44.09%) in F1 score. This confirms that modeling the multi-predicate interactions using RNNs contributes to high-performance, even without syntactic information, by learning contextual information effective for PAS analysis from the word sequence of the sentence.

Effects of Global Features

Table 4.3 shows the effectiveness of the global features on the development set. We

L		Single-Seq		Multi-Seq	
		$+res.$	$-res.$	$+res.$	$-res.$
2	*Dep*	**87.34**	87.10	87.43	**87.73**
	Zero	**47.98**	47.90	**47.66**	46.93
	All	**80.62**	80.24	**80.71**	80.68
4	*Dep*	87.27	**87.41**	**87.60**	87.09
	Zero	50.43	**50.83**	48.10	**48.58**
	All	80.92	**80.99**	**80.99**	80.59
6	*Dep*	**87.73**	87.11	**88.04**	87.39
	Zero	48.81	**49.51**	**48.98**	48.91
	All	**81.05**	80.63	**81.19**	80.68
8	*Dep*	**87.98**	87.23	**87.65**	87.07
	Zero	47.40	**48.38**	**49.34**	48.23
	All	**81.31**	80.33	**81.33**	80.40

Table 4.4: Performance comparison for different numbers of layers on the development set in F1 score. L is the number of the RNN or Grid layers. $+res.$ or $-res.$ indicates whether the model has residual connections (+) or not (−).

incrementally add the global features to the both models that utilize only the local features. The results show that the global features improved the performance by around 1.0% in F1 score overall, i.e., 77.89% to 78.85% in PCJoint and 77.43% to 79.17% in ACJoint. In particular, they are beneficial to zero argument identification (Zero), i.e., an improvement of 1.99% in the Per-Case Joint Model and 3.12% in the All-Cases Joint Model).

Effects of Network Depth

Table 4.4 presents development F1 scores from the neural models with different network depths and with/without residual connections. The performance tends to improve as the RNN or Grid layers get deeper with residual connections. In particular, the two models with eight layers and residual connections achieved considerable improvements of approximately 1.0% according to the F1 scores

	Dep			Zero		
	NOM	ACC	DAT	NOM	ACC	DAT
	NAIST Text Corpus 1.5					
Baseline	86.50	92.84	30.97	45.56	21.38	0.83
PCJoint	87.54	93.09	34.19	47.62	22.73	0.83
ACJoint	88.13	92.74	38.39	48.11	24.43	4.80
SingleSeq	88.32	93.89	65.91	49.51	35.07	9.83
MultiSeq	88.75	93.68	64.38	50.65	32.35	7.52
	NAIST Text Corpus 1.4β					
Taira+ 08*	75.53	88.20	89.51	30.15	11.41	3.66
Imamura+ 09*	87.0	93.9	80.8	50.0	30.8	0.0
Sasano+ 11*	-	-	-	39.5	17.5	8.9

Table 4.5: Performance comparison for different case roles on the test set in F1 score. NOM, ACC or DAT is the nominal, accusative or dative case, respectively. The asterisk (*) indicates that the model uses external resources.

compared to models without residual connections. This means that residual connections contribute to effective parameter learning of deeper models.

Comparison per Case Role

Table 4.5 shows F1 scores for each case role on the test set. For reference, we show the results of the previous studies using the NAIST Text Corpus 1.4β with external resources as well.[6]

Comparing the models using the NAIST Text Corpus 1.5, the single- and multi-sequence models (SingleSeq and MultiSeq) outperformed the other three models (Baseline, PCJoint and ACJoint) according to all metrics. In particular, for direct dependency arguments (Dep) of the dative case (DAT), the

[6]The major difference between the NAIST Text Corpus 1.4β and 1.5 is the revision of the annotation criterion for the dative case (DAT) (corresponding to Japanese case marker "に"). Argument and adjunct usages of the case marker "に" are not distinguished in 1.4β, making the identification of the dative case seemingly easy.

two neural models achieved much higher results, by approximately 30%. This suggests that although dative arguments appear infrequently compared with the other two case arguments, the neural models can learn them robustly. In addition, for zero arguments (ZERO) of the nominative case (NOM), the multi-sequence model demonstrated a considerable improvement of approximately 2.5% according to the F1 scores compared with the bipartite graph models (PCJOINT and ACJOINT). To achieve high accuracy for the analysis of such zero arguments, it is necessary to capture long distance dependencies [27, 53, 30]. Therefore, the performance improvements of zero arguments suggest that the neural models effectively capture long distance dependencies using RNNs that can encode the context of the entire sentence.

4.6 Related Work

4.6.1 Japanese PAS Analysis Approaches

For Japanese PAS analysis research, the NAIST Text Corpus has been used as a standard benchmark [28]. Existing approaches to Japanese PAS analysis are divided into two categories: (i) the *pointwise approach* and (ii) the *joint approach*.

The pointwise approach involves estimating the score of each argument candidate for one predicate, and then selecting the argument candidate with the maximum score as an argument [57, 32, 21, 31]. One of the representative researches is Imamura et al. (2009). They built three distinct models corresponding to the three case roles by extracting features defined on each pair of a predicate and a candidate argument. Using each model, they select the best candidate argument for each case per predicate. Their models are based on maximum entropy model and can easily incorporate various features, resulting in high accuracy.

The joint approach involves scoring all the predicate-argument combinations in one sentence, and then selecting the combination with the highest score [61, 53, 20, 55]. Sasano and Kurohashi (2011) simultaneously determines all the three case arguments per predicate by exploiting large-scale case frames obtained from large raw texts. They focus on identification of implicit arguments (ZERO and INTERZERO), and achieves comparable results to Imamura et al. (2009). Yoshikawa et al. (2011) determines all case arguments for all predicates in a sentence using Markov Logic Networks. Shibata et al. (2016) also simultaneously determines

all predicate-argument combinations using our all-cases joint model with neural networks. Compared with the pointwise approach, their methods based on the joint approach achieve better results.

4.6.2 Modeling of Multi-Predicate Interactions

In semantic role labeling (SRL), Yang and Zong (2014) [60] proposed a model based on the linguistic intuition that the predicates in a sentence are semantically related to each other, and that the information regarding this semantic relation can be useful for SRL. They reported that their reranking model, which captures the multi-predicate interactions, is effective for the English constituent-based SRL task [6]. Taking this a step further, we propose new models capturing interactions between multiple predicates and arguments.

4.6.3 Neural Approaches

Japanese PAS

In recent years, several attempts have been made to apply neural networks to Japanese PAS analysis [55, 31][7]. In [55], a feed-forward neural network is used for the score calculation part of our all-cases joint model. In [31], multi-column convolutional neural networks are used for the zero anaphora resolution task.

Both models exploit syntactic and selectional preference information as the atomic features of neural networks. Overall, the use of neural networks has resulted in advantageous performance levels, mitigating the cost of manually designing combination features. In this thesis, we demonstrate that even without such syntactic information, our neural models can realize strong performance exclusively using the word sequence information of a sentence.

English SRL

Some neural models have achieved high performance without syntactic information in English SRL. [10] and [67] worked on the English constituent-based SRL task [6] using neural networks. In [10], their model exploited a convolutional neural network and achieved a 74.15% F-measure without syntactic information. In

[7]These previous studies used unpublished datasets and evaluated the performance with different experimental settings. Consequently, we cannot compare their models with ours.

[67], their model exploited bidirectional RNNs with linear-chain conditional random fields (CRFs) and achieved the state-of-the-art result, an 81.07% F-measure. Our models should be regarded as an extension of their model.

The main differences between [67] and our work are: (i) constituent-based vs dependency-based argument identification and (ii) the multi-predicate consideration. For the constituent-based SRL, [67] used CRFs to capture the IOB label dependencies, because systems are required to identify the *spans* of arguments for each predicate. By contrast, for Japanese dependency-based PAS analysis, we replaced the CRFs with the softmax function, because in Japanese, arguments are rarely adjacent to each other.[8] Furthermore, whereas the model described in [67] predicts arguments for each predicate independently, our multi-sequence model jointly predicts arguments for all predicates in a sentence concurrently by considering the multi-predicate interactions.

4.7 Summary

In this thesis, we present two types of models: (i) bipartite graph models and (ii) Grid-RNN models.

The bipartite graph models capture interactions between multiple predicates and arguments using a bipartite graph and greedily search the optimal PAS combination in a sentence. Experiments on the NAIST Text Corpus demonstrate that capturing the *multi-predicate interactions* is effective for Japanese PAS analysis. In particular, ZERO argument identification, one of the problematic issues in Japanese PAS analysis, is improved by taking such interactions into account.

The Grid-RNN models automatically induce effective feature representations from the word sequence information using grid-type recurrent neural networks. Experimental results show that the Grid RNN models achieve high performance without the need for syntactic information. Especially, these models improve the performance of ZERO argument identification by considering the *multi-predicate interactions* with Grid-RNNs, which is consistent with the results of our bipartite graph models.

Since our models are applicable to SRL, applying our models for multilingual SRL tasks is an interesting future research direction. Also, in this thesis, the

[8] In our preliminary experiment, we could not confirm the performance improvement by CRFs.

model parameters were learned without any external resources, so that we plan to explore effective methods for exploiting large-scale unlabeled data to learn our models.

Chapter 5

Conclusion

5.1 Summary

This thesis aimed to improve syntactic and semantic dependency parsing.

In Chapter 3, to improve syntactic dependency parsing, we presented a supertag design framework that is flexible so that various supertag sets may be designed. Based on the framework, we instantiated various granularity supertag sets that encode rich syntactic information. To investigate the utility of these supertag features, we have performed the experiments in multilingual dependency parsing. Experimental results show that in order to improve dependency parsing, it is critical to encode the head directionality, head label, and dependent possession information as supertags. In particular, the head label information is crucial for improving LAS. By contrast, the obligatory dependent labels do not improve the results.

In Chapter 4, to improve semantic dependency parsing, we presented two types of approaches using (i) bipartite graphs and (ii) grid-type recurrent neural networks (Grid-RNNs). The first approach represents the interactions between predicates and arguments using a bipartite graph and greedily searches the optimal PAS combination in a sentence. The second approach automatically induces effective feature representations from the word sequence information of a sentence using Grid-RNNs. Experimental results show that capturing the multi-predicate interactions is effective for semantic dependency parsing. In particular, zero argument identification, one of the problematic issues in Japanese PAS analysis, is improved by taking such interactions into account.

5.2 Future Directions

For syntactic dependency parsing, we would like to investigate the interaction of supertag features with higher-order features and explore linguistic entities that capture structurally richer information, such as subtree structures. For semantic dependency parsing, applying our approaches for multilingual settings presents an interesting future research direction. Also, because in this work our models were learned with only labeled data, we plan to explore effective methods for exploiting large-scale unlabeled data to learn the models.

Appendix A

First Appendix

A.1 Feature Templates for Supertagging

A.2 Feature Templates for Dependency Parsers

NAME	FEATURE WINDOW	FEATURE TEMPLATE
UNIGRAM	for p in x_{i-3}, x_{i-2}, x_{i-1}, x_i, x_{i+1}, x_{i+2}, x_{i+3}	$\langle p.t \rangle$, $\langle p.w \rangle$
BIGRAM	for p,q in (x_i,x_{i+1}), (x_i,x_{i+2}), (x_i,x_{i+3}), (x_{i-1},x_i), (x_{i-2},x_i), (x_{i-3},x_i), (x_{i+1},x_{i+2}), (x_{i-2},x_{i-1})	$\langle p.t \circ q.t \rangle$, $\langle p.w \circ q.w \rangle$
HISTORY		$\langle x_{i-1}.stag \rangle$, $\langle x_{i-2}.stag \rangle$, $\langle x_{i-1}.stag \circ x_i.t \rangle$, $\langle x_{i-2}.stag \circ x_i.t \rangle$, $\langle x_{i-2}.stag \circ x_{i-1}.stag \rangle$, $\langle x_{i-2}.stag \circ x_{i-1}.stag \circ x_i.t \rangle$

Table A.1: Feature templates for the supertagging models. The notations used in this table are as follows: feature conjunction= $\circ$; x_i is the i-th word in the sentence; w=word form; t=POS tag; stag=supertag.

NAME	FEATURE WINDOW	FEATURE TEMPLATE
UNIGRAM	for p in s_0, s_1, s_2, b_0, b_1, b_2	$\langle p.t \rangle$, $\langle p.w \rangle$, $\langle p.t \circ p.lc.t \rangle$, $\langle p.t \circ p.rc.t \rangle$, $\langle p.t \circ p.lc.t \circ p.rc.t \rangle$
BIGRAM	for p,q in (s_2,s_1), (s_1,s_0), (s_0,b_0), (b_0,b_1), (b_1,b_2)	$\langle p.t \circ q.t \rangle$, $\langle p.w \circ q.w \rangle$, $\langle p.t \circ q.w \rangle$, $\langle p.w \circ q.t \rangle$, $\langle p.t \circ q.t \circ p.lc.t \circ q.lc.t \rangle$, $\langle p.t \circ q.t \circ p.rc.t \circ q.lc.t \rangle$, $\langle p.t \circ q.t \circ p.lc.t \circ q.rc.t \rangle$, $\langle p.t \circ q.t \circ p.rc.t \circ q.rc.t \rangle$
STRUCTURAL	for p in s_0, s_1, s_2, b_0, b_1, b_2	$\langle dist(p, p.lc) \circ p.t \rangle$, $\langle dist(p, p.rc) \circ p.t \rangle$, $\langle p.nd \circ p.t \rangle$
	for p, q in (s_2, s_1), (s_1, s_0), (s_0, b_0), (b_0, b_1), (b_1, b_2)	$\langle dist(p, q) \rangle$, $\langle dist(p, q) \circ p.t \circ q.t \rangle$
UNISTAG	for p in s_0, s_1, s_2, b_0, b_1, b_2	$\langle p.stag \rangle$
BISTAG	for p, q in (s_2, s_1), (s_1, s_0), (s_0, b_0), (b_0, b_1), (b_1, b_2)	$\langle p.stag \circ q.stag \rangle$, $\langle p.stag \circ q.t \rangle$, $\langle p.t \circ q.stag \rangle$, $\langle p.stag \circ q.w \rangle$, $\langle p.w \circ q.stag \rangle$

Table A.2: Feature templates for the *arc-standard* model. The notations used in this table are as follows: feature conjunction= $\circ$; s_i=i-th word on the top of the stack; b_i=i-th word in the buffer; lc=left-most dependent; rc=right-most dependent; w=word form; t=POS tag; stag=supertag; $dist(p, q)$=word distance between p and q; nd=1 if the word has no dependent, otherwise 0.

Appendix B

Second Appendix

B.1 Feature Templates for Bipartite Graph Models

B.2 Hyper-Parameters for Neural Models

Structure	Name	Description
Diff-Arg	**PAIR**	$\langle\ p_i.\text{rf} \circ p_j.\text{rf} \circ p_i.\text{vo} \circ p_j.\text{vo}\ \rangle$, $\langle\ a_i.\text{ax} \circ a_i.\text{rp} \circ p_i.\text{ax} \circ p_i.\text{vo}\ \rangle$, $\langle\ a_j.\text{ax} \circ a_j.\text{rp} \circ p_j.\text{ax} \circ p_j.\text{vo}\ \rangle$
	TRIANGLE	$\langle\ a_i.\text{ax} \circ a_i.\text{ax} \circ a_i.\text{rp} \circ a_j.\text{rp} \circ p_i.\text{ax} \circ p_i.\text{vo}\ \rangle$, $\langle\ a_i.\text{ax} \circ a_j.\text{ax} \circ a_i.\text{rp} \circ a_j.\text{rp} \circ p_j.\text{ax} \circ p_j.\text{vo}\ \rangle$
	QUAD	$\langle\ a_i.\text{ax} \circ a_j.\text{ax} \circ a_i.\text{rp} \circ a_j.\text{rp} \circ p_i.\text{vo} \circ p_j.\text{vo}\ \rangle$, $\langle\ a_i.\text{ax} \circ a_j.\text{ax} \circ p_i.\text{ax} \circ p_j.\text{ax} \circ a_i.\text{rp} \circ a_j.\text{rp} \circ p_i.\text{vo} \circ p_j.\text{vo}\ \rangle$, $\langle\ a_i.\text{ax} \circ a_j.\text{ax} \circ p_i.\text{rf} \circ p_j.\text{rf} \circ a_i.\text{rp} \circ a_i.\text{rp} \circ p_i.\text{vo} \circ p_i.\text{vo}\ \rangle$
Co-Arg	**BI-PREDS**	$\langle\ a_i.\text{rp} \circ p_i.\text{rf} \circ p_j.\text{rf}\ \rangle$, $\langle\ a_i.\text{ax} \circ a_i.\text{rp} \circ p_i.\text{rf} \circ p_j.\text{rf}\ \rangle$
	DEP-REL	$\langle\ a_i.\text{ax} \circ a_i.\text{rp} \circ p_i.\text{ax} \circ p_j.\text{ax} \circ p_i.\text{vo} \circ p_j.\text{vo} \circ (x, y).\text{dep}\ \rangle$ if x depends on y for x,y in (p_i,p_j), (a_i,p_i), (a_i,p_j), (p_i,a_i), (p_j,a_i)

Table B.1: Global feature templates. p_i, p_j is a predicate, a_i is the argument connected with p_i, and a_j is the argument connected with p_j. Feature conjunction is indicated by $\circ$; ax=auxiliary, rp=relative position, vo=voice, rf=regular form, dep=dependency. All the features are conjoined with the relative position and the case role labels of the two predicates.

Name	Value
Word Embedding Dim d_{word}	32
Mark Embedding Dim d_{mark}	32
Hidden Unit Dim d_{hidden}	32
No. RNN Layers	$\{2, 4, 6, 8\}$
Mini-Batch Size	{ 2, 4, 8 }
Window Size C	5
Learning Rate	Adam[34]
L2 Reg. Coefficient	{ 0.0001, 0.0005, 0.001 }

Table B.2: Hyper-parameters used in the experiments.

Bibliography

[1] B. R. Ambati, T. Deoskar, and M. Steedman. Improving dependency parsers using combinatory categorial grammar. In *Proceedings of EACL*, pp. 159–163, 2014.

[2] S. Bangalore and A. K. Joshi. Supertagging: An approach to almost parsing. *Computational Linguistics*, 25(2):237–265, 1999.

[3] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. J. Goodfellow, A. Bergeron, N. Bouchard, and Y. Bengio. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[4] A. Björkelund, L. Hafdell, and P. Nugues. Multilingual semantic role labeling. In *Proceedings of CoNLL: Shared Task*, pp. 43–48, 2009.

[5] B. Bohnet and J. Nivre. A transition-based system for joint part-of-speech tagging and labeled non-projective dependency parsing. In *Proceedings of EMNLP/COLING*, pp. 1455–1465, 2012.

[6] X. Carreras and L. Màrquez. Introduction to the CoNLL-2005 shared task: Semantic role labeling. In *Proceedings of CoNLL*, pp. 152–164, 2005.

[7] D. Chen and C. Manning. A fast and accurate dependency parser using neural networks. In *Proceedings of EMNLP*, pp. 740–750, 2014.

[8] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of EMNLP*, pp. 1724–1734, 2014.

[9] M. Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of EMNLP*, pp. 1–8, 2002.

[10] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 2011.

[11] M.-C. de Marneffe, T. Dozat, N. Silveira, K. Haverinen, F. Ginter, J. Nivre, and C. D. Manning. Universal Stanford dependencies: A cross-linguistic typology. In *Proceedings of LREC*, pp. 4585–4592, 2014.

[12] M.-C. de Marneffe, B. MacCartney, and C. D. Manning. Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*, pp. 449–454, 2006.

[13] M.-C. de Marneffe and C. D. Manning. The Stanford typed dependencies representation. In *COLING-2008: Proceedings of the Workshop on Cross-Framework and Cross-Domain Parser Evaluation*, pp. 1–8, 2008.

[14] K. Foth, T. By, and W. Menzel. Guiding a Constraint Dependency Parser with Supertags. In *Proceedings of COLING/ACL 2006*, pp. 289–296, 2006.

[15] Y. Goldberg and M. Elhadad. An efficient algorithm for easy-first non-directional dependency parsing. In *Proceedings of HLT/NAACL*, pp. 742–750, 2010.

[16] A. Graves, S. Fernández, and J. Schmidhuber. Bidirectional LSTM networks for improved phoneme classification and recognition. In *Proceedings of International Conference on Artificial Neural Networks*, pp. 799–804, 2005.

[17] A. Graves, N. Jaitly, and A.-r. Mohamed. Hybrid speech recognition with deep bidirectional LSTM. In *Proceedings of Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop*, 2013.

[18] A. Graves and J. Schmidhuber. Offline handwriting recognition with multi-dimensional recurrent neural networks. In *Proceedings of NIPS*, pp. 545–552, 2009.

[19] J. Hajič, M. Ciaramita, R. Johansson, D. Kawahara, M. A. Martí, L. Màrquez, A. Meyers, J. Nivre, S. Padó, J. Štěpánek, et al. The conll-2009 shared task: Syntactic and semantic dependencies in multiple languages. In *Proceedings of CoNLL: Shared Task*, pp. 1–18, 2009.

[20] M. Hangyo, D. Kawahara, and S. Kurohashi. Japanese zero reference resolution considering exophora and author/reader mentions. In *Proceedings of EMNLP*, pp. 924–934, 2013.

[21] Y. Hayashibe, M. Komachi, and Y. Matsumoto. Japanese predicate argument structure analysis exploiting argument position and type. In *Proceedings of IJCNLP*, pp. 201–209, 2011.

[22] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[23] L. He, K. Lee, M. Lewis, and L. Zettlemoyer. Deep semantic role labeling: What works and what ' s next. In *Proceedings of ACL*, 第 1 巻, pp. 473–483, 2017.

[24] L. Huang. Forest reranking: Discriminative parsing with non-local features. In *Proceedings of 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pp. 586–594, Columbus, Ohio, June 2008. Association for Computational Linguistics.

[25] L. Huang, S. Fayong, and Y. Guo. Structured perceptron with inexact search. In *Proceedings of NAACL/HLT*, pp. 142–151, 2012.

[26] L. Huang and K. Sagae. Dynamic programming for linear-time incremental parsing. In *Proceedings of ACL*, pp. 1077–1086, 2010.

[27] R. Iida, K. Inui, and Y. Matsumoto. Anaphora resolution by antecedent identification followed by anaphoricity determination. *ACM Transactions on Asian Language Information Processing (TALIP)*, 4(4):417–434, 2005.

[28] R. Iida, M. Komachi, K. Inui, and Y. Matsumoto. Annotating a Japanese text corpus with predicate-argument and coreference relations. In *Proceedings of the Linguistic Annotation Workshop*, pp. 132–139, 2007.

[29] R. Iida and M. Poesio. A cross-lingual ILP solution to zero anaphora resolution. In *Proceedings of ACL-HLT*, pp. 804–813, 2011.

[30] R. Iida, K. Torisawa, C. Hashimoto, J.-H. Oh, and J. Kloetzer. Intra-sentential zero anaphora resolution using subject sharing recognition. In *Proceedings of EMNLP*, pp. 2179–2189, 2015.

[31] R. Iida, K. Torisawa, J.-H. Oh, C. Kruengkrai, and J. Kloetzer. Intra-sentential subject zero anaphora resolution using multi-column convolutional neural network. In *Proceedings of EMNLP*, pp. 1244–1254, 2016.

[32] K. Imamura, K. Saito, and T. Izumi. Discriminative approach to predicate-argument structure analysis with zero-anaphora resolution. In *Proceedings of ACL-IJCNLP*, pp. 85–88, 2009.

[33] N. Kalchbrenner, I. Danihelka, and A. Graves. Grid long short-term memory. In *Proceedings of ICLR*, 2016.

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

[35] T. Koo, X. Carreras, and M. Collins. Simple semi-supervised dependency parsing. In *Proceedings of ACL/HLT*, pp. 595–603, 2008.

[36] S. Kübler, R. McDonald, and J. Nivre. *Dependency Parsing*. Morgan and Clapool, 2009.

[37] T. Kudo and Y. Matsumoto. Japanese dependency analysis using cascaded chunking. In *Proceedings of CoNLL*, pp. 63–69, 2003.

[38] J. Lafferty, A. McCallum, and F. C. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of ICML*, 2001.

[39] T. Lei, Y. Zhang, L. Màrquez, A. Moschitti, and R. Barzilay. High-order low-rank tensors for semantic role labeling. In *Proceedings of NAACL-HLT*, pp. 1150–1160, 2015.

[40] M. P. Marcus, B. Santorini, and M. Marcinkiewicz. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993.

[41] R. McDonald, J. Nivre, Y. Quirmbach-Brundage, Y. Goldberg, D. Das, K. Ganchev, K. Hall, S. Petrov, H. Zhang, O. Täckström, C. Bedini, N. Bertomeu Castelló, and J. Lee. Universal dependency annotation for multilingual parsing. In *Proceedings of ACL*, pp. 92–97, 2013.

[42] R. McDonald and F. Pereira. Online learning of approximate dependency parsing algorithms. In *Proceedings of the 11th conference on European Chapter of the Association for Computational Linguistics (EACL)*, pp. 81–88, Trento, Italy, April 2006. Association for Computational Linguistics.

[43] A. Nasr and O. Rambow. Supertagging and full parsing. In *Proceedings of the International Workshop on Tree Adjoining Grammar and Related Formalisms (TAG+ 7)*, pp. 56–63, 2004.

[44] J. Nivre. Dependency grammar and dependency parsing. *Technical Report MSI report 05133, Växjö University: School of Mathematics and Systems Engineering*, 2003.

[45] J. Nivre. Incrementality in deterministic dependency parsing. In *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*, pp. 50–57, 2004.

[46] J. Nivre. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34:513–553, 2008.

[47] Y. Nivre. An efficient algorithm for projective dependency parsing. In *Proceedings of IWPT*, pp. 149–160, 2003.

[48] Y. Nivre, J. Hall, and J. Nilsson. Memory-based dependency parsing. In *Proceedings of CoNLL*, pp. 49–56, 2004.

[49] Y. Nivre and M. Scholz. Deterministic dependency parsing of English text. In *Proceedings of COLING*, pp. 64–70, 2004.

[50] M. Palmer, D. Gildea, and P. Kingsbury. The proposition bank: An annotated corpus of semantic roles. *Computational linguistics*, 31(1):71–106, 2005.

[51] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of ICML*, 2013.

[52] S. Petrov, D. Das, and R. McDonald. A universal part-of-speech tagset. In *Proceedings of LREC*, pp. 2089–2096, 2012.

[53] R. Sasano and S. Kurohashi. A discriminative approach to Japanese zero anaphora resolution with large-scale lexicalized case frames. In *Proceedings of IJCNLP*, pp. 758–766, 2011.

[54] M. Schuster and K. K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, pp. 2673–2681, 1997.

[55] T. Shibata, D. Kawahara, and S. Kurohashi. Neural network-based model for Japanese predicate argument structure analysis. In *Proceedings of ACL*, pp. 1235–1244, 2016.

[56] M. Surdeanu, R. Johansson, A. Meyers, L. Màrquez, and J. Nivre. The conll-2008 shared task on joint parsing of syntactic and semantic dependencies. In *Proceedings of CoNLL: Shared Task*, pp. 159–177, 2008.

[57] H. Taira, S. Fujita, and M. Nagata. A Japanese predicate argument structure analysis using decision lists. In *Proceedings of EMNLP*, pp. 523–532, 2008.

[58] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[59] H. Yamada and Y. Matsumoto. Statistical dependency analysis using support vector machines. In *Proceedings of IWPT*, pp. 195–206, 2003.

[60] H. Yang and C. Zong. Multi-predicate semantic role labeling. In *Proceedings of EMNLP*, pp. 363–373, 2014.

[61] K. Yoshikawa, M. Asahara, and Y. Matsumoto. Jointly extracting Japanese predicate-argument relation with markov logic. In *Proceedings of IJCNLP*, pp. 1125–1133, 2011.

[62] D. Zeman. A universal part-of-speech tagset. In *Proceedings of LREC*, pp. 213–218, 2008.

[63] Y. Zhang and S. Clark. A tale of two parsers: Investigating and combining graph-based and transition-based dependency parsing using beam-search. In *Proceedings of EMNLP*, pp. 562–571, 2008.

[64] Y. Zhang and S. Clark. Syntactic processing using the generalized perceptron and beam search. *Computational Linguistics*, pp. 105–151, 2011.

[65] Y. Zhang, T. Lei, R. Barzilay, and T. Jaakkola. Greed is good if randomized: New inference for dependency parsing. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1013–1024, Doha, Qatar, October 2014. Association for Computational Linguistics.

[66] Y. Zhang and J. Nivre. Transition-based dependency parsing with rich non-local features. In *Proceedings of ACL/HLT*, pp. 188–193, 2011.

[67] J. Zhou and W. Xu. End-to-end learning of semantic role labeling using recurrent neural networks. In *Proceedings of ACL-IJCNLP*, 2015.

List of Publications

Journal Papers

1. Hiroki Ouchi, Kevin Duh, Hiroyuki Shindo, and Yuji Matsumoto. "Transition-Based Dependency Parsing Exploiting Supertags". IEEE Transactions on Audio, Speech and Language Processing, Volume: 24, Issue: 11, pp. 2059 - 2068, November 2016.

Conference Papers

1. Hiroki Ouchi, Hiroyuki Shindo, and Yuji Matsumoto. "Neural Modeling of Multi-Predicate Interactions for Japanese Predicate Argument Structure Analysis". In Proceedings of the Association for Computational Linguistics (ACL), pp. 1591 - 1600, July 2017.

2. Hiroki Ouchi, Hiroyuki Shindo, Kevin Duh, and Yuji Matsumoto. "Joint Case Argument Identification for Japanese Predicate Argument Structure Analysis". In Proceedings of the Association for Computational Linguistics (ACL), pp. 961 - 970, July 2015.

3. Hiroki Ouchi, Hiroyuki Shindo, Kevin Duh, and Yuji Matsumoto. "Improving Dependency Parsers with Supertags". In Proceedings of the European Chapter of the Association for Computational Linguistics (EACL), pp. 154 - 158, April 2014.

Awards

1. Outstanding Research Award, Information Processing Society of Japan SIG-NL-233, "Neural Domain Adaptation for Unknown-Domains in Semantic Role Labeling". (in Japanese)

2. Young Researcher Award, Association for Natural Language Processing, “Neural Inter-Sentential Zero-Anaphora Resolution”. (in Japanese)

3. Outstanding Research Award, Information Processing Society of Japan SIG-NL-229, “Deep Recurrent Models for Japanese Predicate Argument Structure Analysis”. (in Japanese)

Other Publication

1. Hiroki Ouchi and Yuta Tsuboi. “Addressee and Response Selection for Multi-Party Conversation”. In Proceedings of the Empirical Methods in Natural Language Processing (EMNLP), pp. 2133 - 2143, November 2016.