Structured pointing networks for natural language understanding

Nguyen, Thanh Tung

2021

Nguyen, T. T. (2021). Structured pointing networks for natural language understanding. Doctoral thesis, Nanyang Technological University, Singapore.
<https://hdl.handle.net/10356/152207>

<https://hdl.handle.net/10356/152207>

<https://doi.org/10.32657/10356/152207>

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License (CC BY-NC 4.0).

Downloaded on 26 Apr 2023 22:26:57 SGT

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

SINGAPORE

**STRUCTURED POINTING NETWORKS FOR
NATURAL LANGUAGE UNDERSTANDING**

NGUYEN THANH TUNG

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

2021

STRUCTURED POINTING NETWORKS FOR NATURAL LANGUAGE UNDERSTANDING

NGUYEN THANH TUNG

School of Computer Science and Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfilment of the requirements for the degree of
Doctor of Philosophy

2021

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

12/05/2021

Date

A handwritten signature in black ink, appearing to read 'Tung', with a horizontal line underneath.

Nguyen Thanh Tung

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

12/05/2021

Date

Asst. Prof. Shafiq Rayhan Joty

Authorship Attribution Statement

This thesis contains materials from four papers published in the following peer-reviewed conferences in which I am listed as a first author.

[Chapter 3](#) is published with material from: Efficient Constituency Parsing by Pointing. Thanh-Tung Nguyen, Xuan-Phi Nguyen, Shafiq Joty, Xiaoli Li. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL 2020)*, pages 3284-3294, Virtual.

The contributions of the co-authors are as follows:

- I came up with the key idea, designed all experiments, implemented all of the source code and conducted all experiments. I prepared the manuscript drafts (in entirety).
- The manuscript was revised and edited mainly by Assist. Prof. Shafiq Joty, Xuan-Phi Nguyen and Dr. Xiaoli Li.

[Chapter 4](#) and a portion of [Chapter 5](#) (RST parsing at the sentence level) are published with material from: A Conditional Splitting Framework for Efficient Constituency Parsing. Thanh-Tung Nguyen, Xuan-Phi Nguyen, Shafiq Joty, Xiaoli Li. *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics (ACL 2021)*, Virtual.

The contributions of the co-authors are as follows:

- I came up with the key idea, designed all experiments, implemented all of the source code and conducted all experiments. I prepared the manuscript drafts (in entirety).
- The manuscript was revised and edited mainly by Assist. Prof. Shafiq Joty, Xuan-Phi Nguyen and Dr. Xiaoli Li.

A part of [Chapter 5](#) (RST parsing at the document level) is published with material from: RST Parsing from Scratch. Thanh-Tung Nguyen, Xuan-Phi Nguyen, Shafiq Joty, Xiaoli Li. *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (NAACL 2021)*, Virtual.

The contributions of the co-authors are as follows:

- I came up with the key idea, designed all experiments, implemented all of the source code and conducted all experiments. I prepared the manuscript drafts (in entirety).
- The manuscript was revised and edited mainly by Assist. Prof. Shafiq Joty, Xuan-Phi Nguyen and Dr. Xiaoli Li.

Chapter 6 is published with material from: Differentiable Window for Dynamic Local Attention. Thanh-Tung Nguyen, Xuan-Phi Nguyen, Shafiq Joty, Xiaoli Li. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL 2020)*, pages 6589-6599, Virtual.

The contributions of the co-authors are as follows:

- I came up with the key idea, designed all experiments, implemented all of the source code and conducted all experiments. I prepared the manuscript drafts (in entirety).
- The manuscript was revised and edited mainly by Assist. Prof. Shafiq Joty, Xuan-Phi Nguyen and Dr. Xiaoli Li.

12/05/2021

Date

Nguyen Thanh Tung

Acknowledgements

I would like to express my sincere gratitude towards my supervisors, Assist. Prof. Shafiq Joty and Dr. Xiaoli Li, for providing invaluable guidance and encouragement throughout my PhD journey. They inspire me with many valuable lessons, amongst of which help to enhance my passion to discover new problems and solutions. They are not only my supervisors but also my friends, who are always there to support me both mentally and technically. It has been my pleasure to work under my supervisors' mentorship.

In addition, I am thankful to my friends and also collaborators whom I have learned so much from. Firstly, I would like to express my gratitude to Xuan-Phi Nguyen and Tay Yi for their inestimable support and motivation to help me to complete my work. I would also like to thank my other colleagues and friends in the NTU-NLP group including Jwala, Han Cheol, Lin Xiang, Simeng, Maruf and Tasnim for their discussion and inspiration. I feel very lucky and appreciate to have them as friends and as part of my PhD journey. Last but not least, I would like to thank my family for their unconditional support and encouragement, without which this work would have not been possible.

Abstract

Natural Language Understanding (NLU) is a subfield in Natural Language Processing (NLP) that deals with machine comprehension of the structure and meaning of human languages. Generally, NLU is the core component of all NLP applications. Therefore, unlocking NLU would help to improve these applications into human parity by extracting more useful information from textual data. However, it is very challenging because understanding natural language requires utilizing the relevant language context to get the true meanings of the text and how the components of the text are connected with each other to form the right structure of the language. Notably, NLU involves finding such linguistic structures and contextual interpretation not only at the sentence level but also at the document (discourse) level.

With the huge success of deep learning, computers can be taught to understand human languages by calculating the relative importance among the inputs to find the contextual representation, to point to the relevant part in the input and then performing inference on those representations. With this advance in technology, this dissertation tackles the problem of NLU in two aspects: building better language tools to analyze natural language in terms of explicit representations of the syntax and discourse structures and developing better less-human-effort language modules for many downstream problems. In the former aspect, this thesis generally transforms the complex structures into easier tree representations and propose more efficient parsers to learn and decode the new representations. In the latter aspect, we propose neural modules that utilize an inductive bias to capture the context of language more comprehensively. In particular, this dissertation proposes:

- **A Span-based Pointing Parser** ([Chapter 3](#)), a novel constituency parsing model that casts the parsing problem into a series of span-based pointing tasks. Specifically, our model estimates the likelihood of a span being a legitimate tree constituent via

the pointing score corresponding to the boundary words of the span. Our parsing model supports efficient top-down decoding and our learning objective is able to enforce structural consistency without resorting to the expensive structured inference (e.g., CKY). The experiments on the standard English Penn Treebank parsing task show that our method achieves the higher accuracy than all the existing methods with similar time complexity. Using pre-trained embeddings, our model achieves competitive results with the state-of-the-art while being faster. Our approach also establishes new state-of-the-art in some multilingual constituency parsing tasks.

- **A Conditional Splitting Parser** (Chapter 4), a generic seq2seq parsing framework that casts constituency parsing problems into a series of conditional splitting decisions. Our parsing model estimates the conditional probability distribution of possible splitting points in a given text span and supports efficient top-down decoding, which is linear in number of nodes. The conditional splitting formulation together with efficient beam search inference facilitate structural consistency without relying on expensive structured inference. Experiments show that our model achieves good results on the standard syntactic parsing tasks under settings with/without pre-trained representations and rivals state-of-the-art (SoTA) methods that are more computationally expensive than ours. It also outperforms the span-based pointing parser mentioned above.
- **An End-to-end RST Parser** (Chapter 5), a novel top-down end-to-end formulation of sentence and document level discourse parsing in the Rhetorical Structure Theory (RST) framework. In this formulation, we consider discourse parsing as a sequence of splitting decisions at token boundaries and use a seq2seq network to model the splitting decisions. Our framework facilitates discourse parsing from scratch without requiring discourse segmentation as a prerequisite; rather, it yields segmentation as part of the parsing process. Our unified parsing model adopts a beam search to decode the best tree structure by searching through a space of high scoring trees. With extensive experiments on the standard RST discourse treebank, we demonstrate that our parser outperforms existing methods by a good margin in both end-to-end parsing and parsing with gold segmentation. More importantly, it

does so without using any handcrafted features, making it faster and easily adaptable to new languages and domains.

- **A Dynamic Differentiable Attention Window** ([Chapter 6](#)), a new neural module and general purpose component for dynamic window (span) selection. While universally applicable, we demonstrate a compelling use case of utilizing Differentiable Window to improve standard attention modules by enabling more focused attentions over the input regions. We propose two variants of Differentiable Window, and integrate them within the Transformer architecture in two novel ways. We evaluate our proposed approach on a myriad of NLP tasks, including machine translation, sentiment analysis, subject-verb agreement and language modeling. Our experimental results demonstrate consistent and sizable improvements across all tasks.

In this dissertation, we conducted extensive experiments of all models across several highly popular benchmark datasets. We also conducted qualitative and visualization experiments to understand the inner workings of our proposed models. This dissertation concludes with some potential directions that we plan to investigate in the future.

Contents

Acknowledgements	v
Abstract	vi
List of Figures	xiii
List of Tables	xvi
Acronyms	xviii
1 Introduction	1
1.1 Background	1
1.1.1 Hierarchical Structures of Languages	2
1.1.2 Context-aware Language Representations	5
1.1.3 Summmary	6
1.2 Research Problems and Methodologies	6
1.2.1 Effective and Efficient Constituency Parsing	7
1.2.2 RST Parsing from Scratch	8
1.2.3 Sequential Neural Network with Inductive Bias	9
1.3 Thesis Outline	10
2 Literature Review	11
2.1 Neural Networks for Sequences	11
2.1.1 Recurrent Neural Networks	11
2.1.2 Neural Attention Networks	13
2.2 Pointing Mechanism	17
2.3 Constituency Parsing	19

2.3.1	Chart-based Parsers	21
2.3.2	Transition-based Parsers	22
2.3.3	Other Parsers	23
2.3.4	Evaluation Metrics	24
2.4	Discourse Parsing	25
2.4.1	Chart-based Parsers	27
2.4.2	Transition-based Parsers	28
2.4.3	Other Parsers	29
2.4.4	Evaluation Metrics	30
3	Efficient Constituency Parsing by Pointing	32
3.1	Introduction	32
3.2	Related Work	34
3.3	Proposed Models	35
3.3.1	Parsing as Pointing	35
3.3.2	Top-Down Tree Inference	39
3.3.3	Model Architecture	42
3.3.4	Training Objective	44
3.4	Experiments	44
3.4.1	English (PTB) Experiments	45
3.4.2	SPMRL Multilingual Experiments	48
3.5	Summary	49
4	A Conditional Splitting Framework for Efficient Constituency Parsing	50
4.1	Introduction	50
4.2	Related Work	52
4.3	Parsing as a Splitting Problem	53
4.4	Seq2Seq Parsing Framework	55
4.4.1	Model Architecture	56
4.4.2	Top-Down Beam-Search Inference	58
4.5	Experiments	60
4.5.1	English (PTB) Constituency Parsing	60
4.5.2	SPMRL Multilingual Constituency Parsing	62
4.5.3	Parsing Speed Comparison	63

4.6	Summary	63
5	RST Parsing from Scratch	65
5.1	Introduction	65
5.2	Related Work	68
5.3	Model	68
5.3.1	Parsing as a Splitting Decision Problem	70
5.3.2	Model Architecture	72
5.3.3	Complete Discourse Parsing Models	75
5.4	Experiments	78
5.4.1	Document Parsing with Gold Segmentation	78
5.4.2	Document End-to-end Parsing	80
5.4.3	Ablation Study	81
5.4.4	Sentence Parsing	82
5.4.5	Parsing Speed	83
5.5	Summary	85
6	Differentiable Window for Dynamic Local Attention	86
6.1	Introduction	87
6.2	Related Work	89
6.2.1	Transformer and Global Attention	89
6.2.2	Windows in Attentions	90
6.3	Dynamic Differential Window	91
6.3.1	Discrete Window Masking	91
6.3.2	Trainable Soft Masking	93
6.3.3	Segment-Based Soft Masking	94
6.4	Dynamic Window Attention Methods	95
6.4.1	Multiplicative Window Attention	96
6.4.2	Additive Window Attention	96
6.4.3	Implementation in Transformer	97
6.5	Experiment	98
6.5.1	Machine Translation	98
6.5.2	Ablation Study	100
6.5.3	Text Classification	103

6.5.4	Language Modeling	104
6.6	Summary	105
7	Conclusion	106
7.1	Summary	106
7.2	Future Directions	107
	List of Publications	109
	References	111

List of Figures

1.1	A constituency parse tree for the sentence “He eats spaghetti with meat”. Nodes in the constituency tree are phrasal constituents; <i>e.g.</i> , “spaghetti with meat” corresponds to the phrasal node <i>NP</i>	3
1.2	Example of a dependency tree. Nodes in the dependency tree are words which are directly connected to each other with dependency links. The word “eats” corresponds to the root of the tree or the head of the whole structure.	3
1.3	A discourse tree for two sentences. In this example, each sentence is an EDU, and <i>Explanation</i> is the coherence relation between the nucleus (the second sentence) and a satellite (the first sentence).	4
2.1	An example of Long Short-Term Memory Network. Taken from [1].	12
2.2	An example of computing output of vanilla attention. Taken from [1].	14
2.3	An example of cross attention for Machine Translation. Taken from [2].	15
2.4	An example of encoder self attention for the word “it” in different context. Taken from https://bit.ly/2kDol2S	15
2.5	Architecture of Transformer Network. Taken from [3].	16
2.6	Vanilla pointing network with finding convex hull problem. Taken from [4]. . . .	19
2.7	A parse tree for “She enjoys playing tennis.”	20
2.8	An example of computing a span score. Taken from [5].	22
2.9	An example of the transition sequence of a parse tree. Taken from [6].	23
2.10	An example of a predicted tree and its gold tree.	24
2.11	An example of an evaluation on constituency parsing	25
2.12	Example of a discourse tree. Taken from [7].	26
2.13	An example of a chart-based discourse parser. Taken from [8].	28
2.14	An example of a discourse tree’s transition operation sequence (source[9]). . . .	28

2.15	An example of a constituent-based evaluation on discourse parsing. Taken from [10]	30
3.1	A binarized constituency tree for the sentence “She enjoys playing tennis.”. The node $S-VP$ is an example of a collapsed atomic label. We omit POS tags and singleton spans for simplicity. Below the tree, we show span and pointing representations of the tree.	36
3.2	Inferring the parse tree for a given sentence and its part-of-speech (POS) tags (predicted by an external POS tagger). Starting with the full sentence span $(1, 5)$ and its label S , we predict split point 1 using the base (sp) and general (gp) pointing scores as per Eqn. 3.2-3.4. The left singleton span $(1, 1)$ is assigned with a label NP and the right span $(2, 5)$ is assigned with a label $\emptyset$ using the label classifier gc as per Eqn. 3.5. The recursion of splitting and labeling continues until the process reaches a terminal node. The label assignment for the unary spans is done by the uc classifier. . . .	41
4.1	An binarized constituency of the sentence “She enjoys playing tennis.”. Below the tree, we show the labeled span and splitting representations.	53
4.2	Our constituency parser along with the decoding process for a given sentence. The input to the decoder at each step is the representation of the span to be split. We predict the splitting point using a biaffine function between the corresponding decoder state and the boundary-based encoder representations. A label classifier is used to assign labels to the left and right spans.	55
4.3	Illustration of our boundary-based span encoder. Here we have shown the representation for the boundary at 1 and the representation of the boundary-based span $(0, 5)$ that corresponds to the sentence “She enjoys playing tennis.”.	57
4.4	An error example where our system incorrectly labels a NP as $\emptyset$	62
5.1	A discourse tree for two sentences in the RST discourse treebank. The internal nodes (e.g., <i>Attribution</i> , <i>Contrast</i>) denote the coherence relations and the edge labels reflect the nuclearity of the child span. Below the tree, we show the sequence of splitting decisions $\mathbb{S}_{\text{edu}}$ when EDUs are provided and $\mathbb{S}$ when EDUs are not provided (end-to-end parsing). The bold splitting decision represents the final split of the span, forming an EDU.	69

5.2	Relation between token-boundary (above) and token (below) representations. A token-boundary position k is located between the tokens at k and $k + 1$	70
5.3	Our discourse parser along with a few decoding steps for a given document. The input to the decoder at each step is the representation of the span to be split. We predict the splitting point using the biaffine function between the corresponding decoder state and the token-boundary encoder representations. The figure is for end-to-end parsing, where each EDU-corresponding span points to its right edge to mark the EDU. The coherence relations between the left and right spans are assigned using a label classifier after the (approximately) optimal tree structure is formed using beam search.	71
5.4	Illustration of token-boundary span encoder. The figure lays out an example representation for the boundary at 2 and the representation of the token-boundary span $(0, 44)$, which corresponds to the whole document.	73
5.5	Confusion matrix for relation labels on the RST-DT test set. The vertical axis represents true and horizontal axis represents predicted relations. The relations are: Joint (JO), Elaboration (EL), Temporal (TEM), Contrast (CONT), Cause (CA), Same-Unit (SA), Manner-Means (MA), Attribution (AT), Condition (COND), Comparison (COMP), Background (BA), Enablement (EN), Explanation (EX), Evaluation (EV), Summary (SU), Topic-Comment (T-CM),	82
5.6	An error example where our system incorrectly labels a Summary as Elaboration.	82
5.7	Our system incorrectly labels a Condition as Background.	83
5.8	Our system incorrectly labels a Temporal as Joint.	83
5.9	Our system incorrectly labels a Explanation as Elaboration.	83
6.1	Example of ϕ , $\mathbf{f}$, and $\mathbf{g}$ vectors and how the mask vector $\mathbf{m}_q$ can be derived for $l_q = 3$ and $r_q = 8$	92
6.2	Segment-based masking for segment size = 2. Instead of pointing to the left and right indices of the <i>tokens</i> , the soft segment-based method (approximately) points to the left and right boundaries of the <i>segments</i> , respectively.	96
6.3	Visualization of masking scores, and attention scores for our and the original Transformer models.	102

List of Tables

3.1	Results for single models (no pre-training) on the PTB WSJ test set, Section 23.	46
3.2	Results on PTB WSJ test set with pretraining.	47
3.3	Parsing speed for different models computed on the PTB WSJ test set. . . .	47
3.4	SPMRL Multilingual dataset split.	48
3.5	SPMRL experiment single model test.	48
3.6	SPMRL experiment pre-trained model test (with pretraining).	49
4.1	Results for single models (no pre-training) on the PTB WSJ test set, Section 23.	61
4.2	Results on PTB WSJ test set with pretraining.	61
4.3	Results on SPMRL test sets without pre-training. The sign ⁺ denotes that systems use predicted POS tags.	62
4.4	Speed comparison of our parser with existing constituency parsers.	63
5.1	Parsing results with gold segmentation. The sign ⁺ denotes that systems use handcrafted features such as lexical, syntactic, sentence/paragraph boundary features and so on, * denotes that systems use external cross-lingual features and § means that systems use pretrained models.	79
5.2	End-to-end parsing performance.	80
5.3	Ablation test of our models <i>with</i> gold EDUs. W/o Beam search indicates the full model with greedy decoding (beam width 1), while w/o Boundary LSTM is the full model with greedy decoding and no LSTM connection between EDU-boundary representations.	81
5.4	Ablation test of our end-to-end model. W/o GloVe is the full model with randomized word embeddings, while w/o Sentence guidance is the full model with randomized word embeddings and without sentence guidance.	81

5.5	Results on sentence-level discourse parsing tasks on the RST-DT test set with and without gold segmentation.	84
5.6	Speed comparison of our parser with existing parsers at sentence level. . .	84
5.7	The wall time for parsing the RST-DT test set at document level.	85
6.1	BLEU scores for different models in WMT’14 English-German and English-French translation tasks.	100
6.2	Evaluation of Additive Window (AW) and Multiplicative Window (MW) attentions in encoder/decoder self attention and cross attention for full vs. partial settings.	100
6.3	BLEU scores for token- and segment-based masking in cross attention and encoder self-attention. The decoder self-attention always uses token-based masking.	101
6.4	Training efficiency and size of similar models	102
6.5	Classification accuracy on Stanford Sentiment Treebank (SST) and IMDB sentiment analysis and Subject-Verb Agreement(SVA) tasks.	103
6.6	Perplexity scores on 1-billion-word language modeling benchmark (the lower the better).	104

Acronyms

NLU	Natural Language Understanding
NLP	Natural Language Processing
CKY	Cocke–Younger–Kasami
SoTA	State-of-the-art
RST	Rhetorical Structure Theory
AI	Artificial Intelligence
NLG	Natural Language Generation
IVR	Interactive Voice Response
DT	Discourse Tree
EDU	Elementary Discourse Unit
RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory
GRU	Gated Recurrent Unit
Seg2seg	Sequence-to-sequence
PCFG	Probabilistic Context Free Grammar
NP	Noun Phrase
Det	Determiner
CRF	Conditional Random Field
MLP	Multi-layer Perceptron
DRT	Discourse Representation Theory
LTAG	Lexicalized Tree Adjoining Grammar
HILDA	High-level Discourse Analyzer

POS	Part-of-speech
FFN	Feed Forward Network
PTB	Penn Treebank
WSJ	Wall Street Journal

Chapter 1

Introduction

1.1 Background

Given the huge growth of the Internet in recent years, people all over the world are communicating increasingly frequently. After all, the quest to understand and process these textual data is at the center of many different applications across various domains. In order to make machines to liaise with human effectively, we must teach them to understand our language or “natural language”. In this thesis, we refer “natural language” as a language used for daily communication by human beings; languages like English, French or Vietnamese. Unlike mathematical notations and programming languages which are created by human with explicit rules, natural languages have evolved through multiple generations and it is difficult for machines to disambiguate the meanings of linguistic units such as words, phrases and sentences close to the human parity.

Natural Language Processing (NLP) is a branch of Artificial Intelligence (AI) that deals with the interaction between machines and humans using natural languages. It includes a wide range problems from simply counting word occurrences to complex ones like translation, summarization, question answering and dialogue analysis. NLP can be broadly split into two subfields Natural Language Understanding (NLU) and Natural Language Generation (NLG). NLU deals with comprehension by machines of the structure and meaning of human languages, while NLG is the process of producing meaningful phrases and sentences in the form of natural language. NLU is an essential and integral part of NLG, which provides relevant information to the generation models. For that reason, NLU has

remained an active research area. NLU plays an important driving force behind the common applications such as language translation applications like Google Translate, Interactive Voice Response (IVR) applications used in call centers to respond to users' requests, and personal assistant applications like Siri, Cortana, Alexa.

One of the difficulties in NLU development is to understand the rules of the human languages. It is not easy for computers to comprehend them efficiently as humans. Some of these rules can be high-level and abstract; for example, when a person uses a sarcastic remark to send a message or when someone refers to an entity mentioned before in the discourse. Moreover, some of these rules can be low-level; for example, using the character "s" to signify the plurality of items or adding the characters "ed"/"ing" to determine when the action took place. Comprehensive understanding of the human language requires understanding both the words and how the concepts are connected in a sentence or more broadly in a discourse to deliver the intended message.

1.1.1 Hierarchical Structures of Languages

To facilitate understanding of human languages, researchers have proposed a number of linguistic formalisms both at sentence level and at document level. At the sentence level, NLU often uses syntax to derive the structured representations of the text. In this thesis, we focus on constituency tree, one of the most popular syntactic structures. Constituency trees combine words into phrases, then phrases into larger phrases. [Fig. 1.1](#) shows a sample constituency tree. In this example, the **noun phrase** "spaghetti with meat" specifies the kinds of spaghetti in this context, as opposed to "he eats meat and spaghetti".

Another popular formalism is dependency structure. It connects words by directed and labeled arcs. Each node in the dependency tree represents a word in the sentence. An arc's label reflects the relation between its start word (head) and its end word (dependent). This dependency structure can uncover the important information that could not be easy to see in the phrase structure tree. For example, [Fig. 1.2](#) shows a dependency tree for the same sentence as the constituency tree example. The argument to the verb "eats" (*i.e.*, "spaghetti") is directly connected to it in the dependency tree.

FIGURE 1.1: A constituency parse tree for the sentence “He eats spaghetti with meat”. Nodes in the constituency tree are phrasal constituents; *e.g.*, “spaghetti with meat” corresponds to the phrasal node *NP*.

FIGURE 1.2: Example of a dependency tree. Nodes in the dependency tree are words which are directly connected to each other with dependency links. The word “eats” corresponds to the root of the tree or the head of the whole structure.

Just like words in a sentence, sentences in a document are also logically related to form a coherent discourse. For instance, let us consider the text “Some fans protested at the stadium. The game was postponed.”. It is clear the first sentence explains the second sentence, which makes the text comprehensible as a whole. It would be otherwise difficult to comprehend it if we replace the second sentence by a random sentence like “I go to sleep early.”. Many theories have been proposed to explain the coherence structure of the text. These include coherence relations based on lexical cohesion [11], Intentional Structure Theory [12], Rhetorical Structure Theory (RST) [13], Centering Theory [14], Discourse Representation Theory (DRT) [15], and Discourse-lexicalized Tree Adjoining Grammar (DLTAG) theory [16].

In this thesis, we focus on RST, which is one of the most widely adopted formalisms in NLP. RST postulates a hierarchical structure of the text called discourse tree (DT). The

leaves of a DT are clause-like units, known as elementary discourse units (EDUs). Adjacent EDUs and higher-order spans are connected hierarchically through coherence relations (e.g., *Explanation*) and their relative importance (e.g., *nuclei* as the main unit, *satellite* as the support unit). Fig. 1.3 shows an example discourse tree for two sentences. The first sentence “Some fans protested at the stadium.” acts as a satellite that helps to **explain** the second (nucleus) sentence “The game was postponed.”

FIGURE 1.3: A discourse tree for two sentences. In this example, each sentence is an EDU, and *Explanation* is the coherence relation between the nucleus (the second sentence) and a satellite (the first sentence).

The Penn Tree Discourse Bank (PDTB) [17] is the another discourse framework. Unlike RST, which connects the adjacent EDUs with coherence relations to build a tree for the entire text, PDTB parsing only focuses on span-pair relation classification, ignoring the hierarchical structure. The way PDTB considers coherence relations is also different from RST. It marks the discourse connectives (e.g., *because*, *however*) which are the words that indicate the type of coherence relations. These connectives can be explicit (in the text) or implicit (not in the context). We show examples of these two cases below:

- *The city’s Campaign Finance Board has refused to pay Mr. Dinkins \$95,142 in matching funds because his campaign records are incomplete.*
- *Second, they channel monthly mortgage payments into semiannual payments, Implicit = so **reducing the administrative burden on investors.***

The underlined word indicates the discourse connective. In the first sentence, the connective is “because”, which indicates an explicit relation “cause” between the second clause and the first clause. In the second sentence, the event in the second clause can be considered as a consequence of the event in the first clause. This sentence does not have a discourse connective, which means the coherence relation is implicit.

While being an interesting research direction, PDTB parsing is considered as *shallow* parsing as it does not provide a hierarchical structure of the text. This makes it less attractive than RST parsing that gives us the ability to model the text as a whole.

NLU entails applying algorithms to identify and extract the natural language rules to convert unstructured texts into effective representations that computers can understand. It is important for the algorithms to be able to capture the syntactic information in the sentence to understand the bigger things from knowing about smaller parts (*i.e.*, the constituency structure) as well as the coherence information underneath the text to know how the discourse units are connected with each other to form a discourse (*i.e.*, the discourse structure). In this thesis, we address the constituency and discourse parsing tasks which are important in capturing the explicit hierarchical structures of the text. Our approach which utilizes pointing mechanism is designed to simplify the problems to make the learning and inference process faster while achieving high accuracy.

1.1.2 Context-aware Language Representations

Learning the explicit representations above may require significant human effort and expense to provide guidance in addition to the original problems (*e.g.*, machine translation, summarization). Therefore, it may be more convenient to build the models that rely minimally on the explicit representations. To achieve a good NLU system with that objective, computers need to understand the true meanings of text and not just the surface interpretation. In other words, machines shall not only recognize the meaning of words but also figure out the context of the data. For example, consider the following sentences:

- The chicken cannot cross the street because it was too tired.
- The chicken cannot cross the street because it was too wide.

The two sentences are slightly different. It is clear that “it” refers to “chicken” in the first sentence and “street” in the second sentence. To understand this coreference, the machine needs to effectively capture the contextual information. This thesis is also concerned with considering new inductive biases to incorporate more local information to current approaches to improve the contextual representation of the data.

1.1.3 Summmary

In this thesis, we tackle NLU with two approaches. One is to develop more effective and efficient parsers to build explicit structures of language at sentence and document level. The other is to introduce better inductive biases to enhance the machines' ability to capture the relevant contexts for language processing tasks. Specifically, for the former, at the sentence level, we propose two novel parsing methods that transform the constituency tree into pointing decisions and infer the structure effectively and efficiently. At the document level, we propose a more comprehensive discourse parser that can build the discourse tree from scratch. To that end, we introduce four novel methods using pointing mechanism for these problems.

1.2 Research Problems and Methodologies

As mentioned above, NLU is challenging because teaching machines to interpret the real context of what is going on at human level is not far from rocket-science. For any task that requires NLU, while it is possible to consider the semantic content of texts to be limited to a narrow field, the accuracy of the NLU algorithm can be much higher if a more general context is considered, so it is important to match the context representation to the desired outcome.

In the past decades, NLU research has primarily focused on either feature-based machine learning [18, 19] or structure-based representations learning [20]. However, both methods still have a discrepancy between features and output (handcrafted features vs task-specific features) which makes them suboptimal. Today deep learning methods dominate NLU systems, thanks to their ability to learn feature automatically with a little human intervention and powerful learning capabilities. In this thesis, we design and develop effective neural models for parsing tasks and multi-purpose neural modules via pointing techniques. The research problems and methodologies are summarized as follows:

1.2.1 Effective and Efficient Constituency Parsing

Recently, like most other NLU tasks, constituency parsing development has advanced with the growth of deep learning. Different neural approaches to this parsing task can be evaluated based on two criteria: the tree representation of the parsing model and the inference algorithm to construct the constituency tree. The transition-based parsers [21–23] represent a constituency tree as a sequence of shift-reduce operations and use language modeling or sequence-to-sequence based methods to train and infer the tree. Therefore, they may suffer from exposure bias (*i.e.*, the train-test discrepancy when the models use the whole ground-truth in training but infer one step at a time in testing [24]). The span-based neural parsers [25–28] represent a constituency tree as a list of labelled spans and learn scoring functions for spans and perform CKY algorithm to find the highest scoring tree for a sentence. These approaches not only require structured inference during training but also suffer from the slow inference time like the non-neural methods.

Approaches and Methodology We propose two novel parsing approaches that represent constituency parsing into a series of pointing decisions. The first approach models the likelihood of the span being a subtree in the constituency tree as the pointing score from one of its endpoints to another. While being a span-based parser, our approach overcomes many disadvantages of this type: our model can be parallelly fully trained without requiring structured inference and also allows efficient top-down decoding with the best and worse case running times of $\mathcal{O}(n \log n)$ and $\mathcal{O}(n^2)$, respectively. The second approach estimates the conditional probability distribution of possible splitting points in a given text span. The conditional splitting formulation together with efficient top-down beam search inference facilitate structural consistency without relying on expensive structured inference. Experiments show that our two models achieve outperforming results on the standard constituency parsing tasks under settings with/without pre-trained representations and rivals state-of-the-art (SoTA) methods that require more expensive computational cost than ours.

1.2.2 RST Parsing from Scratch

As described earlier, RST represents a text by a discourse tree (DT), where the leaves are the elementary discourse units (EDU), and adjacent EDUs and higher-order spans are connected hierarchically through coherence relations and relative importance of nuclearity (As shown in Fig. 1.3). Discourse analysis in RST often involves two subtasks: breaking the text into a sequence of EDUs (*a.k.a.* discourse segmentation) and relating the discourse units into a nested labeled tree (*a.k.a.* discourse parsing). Discourse segmentation has been treated as a separated and prerequisite task for discourse parsing. Similar to constituency parsing, discourse parsing also benefits from using rich feature representation from neural network instead of traditional statistical features. However, due to the task’s nature of dealing with extremely long textual data at document level, successful discourse parsers still depend on handcrafted features [29–32]. Therefore, this expensive feature engineering step makes it difficult to adapt to the new domains/tasks and also affects the training/inference speed.

Approaches and Methodology We propose a novel discourse parsing approach that could generate a discourse tree from scratch without requiring discourse segmentation as a prerequisite step. In other words, our parser performs discourse segmentation as a by-product of parsing and therefore solves the two tasks in a single neural network model. Our approach represents discourse parsing as a sequence of splitting decisions at token boundaries and uses a seq2seq pointer network to model these decisions. Importantly, our parser can adopt beam search to widen the search space for the highest scoring tree. This makes our parser more capable than other parsers, which can only perform greedy decoding. Our model also does not need any handcrafted feature, which allows it to overcome the above mentioned issues. Experiments show that our model obtains good results on the discourse parsing tasks under settings with/without gold EDU segmentation and achieves SoTA results.

1.2.3 Sequential Neural Network with Inductive Bias

Like other fields in AI, NLP has gone through a deep-learning tsunami with neural network models playing essential part because such models learn automatically from the data and require less human guidance. Among all the recent break-throughs in neural networks, computing relative importance among inputs can be regarded as one of the significant advances in modern deep learning research. This paradigm, commonly known as *attention* [33], has demonstrated immense success across a wide spectrum of applications. To this end, learning to compute contextual representations [3], to point to the relevant part in the input [34], or to select windows or spans [35] from sequences forms the crux of many modern deep neural architectures. Despite advances in developing neural modules for computing relative relevance [2, 36], there has been no general purpose solution for learning differentiable attention windows. While span selection-based pointer network models typically predict a start boundary and an end boundary [35, 37], these soft predictions generally reside at the last layer of the network and are softly optimized.

Approaches and Methodology Rather than defining attentions based on prior information like tree structure or fixed local regions, we focus on attentions over automatically learned local windows (*i.e.*, ngrams of tokens) in our work. To the best of our knowledge, there exists no general purpose component for learning differentiable windows within the networks. We propose a novel method to enhance attentions with differentiable windows. By aggregating or limiting attentions to a dynamic local window, the noisy alignment from long input could be eliminated and therefore enhances the attentions. We make several key technical contributions. First, we formulate the dynamic window selection problem as a problem of learning a discrete mask (*i.e.*, binary values representing the window). By learning and composing left and right boundaries, we show that we are able to parameterize the (discrete) masking method. We then propose soft adaptations of the aforementioned, namely trainable soft masking and segment-based soft masking, which are differentiable approximations that can be easily optimized in an end-to-end fashion as well as inherit the desirable properties of discrete masking.

1.3 Thesis Outline

[Chapter 1](#) presents the scope and overview of this dissertation.

[Chapter 2](#) provides the readers with a review of related work in constituency parsing, discourse parsing and multi-purpose context modeling neural modules.

[Chapter 3](#) starts to investigate the constituency parsing task. We propose a pointing span-based neural parser and verify its effectiveness on English and multilingual constituency parsing datasets. The key idea is representing the constituency tree as a sequence of span-based pointing decisions that allows efficient training and inference strategies.

[Chapter 4](#) studies a conditional splitting framework in which we propose a deep neural network model to estimate the conditional probability distribution of possible splitting points in a given text span. The key idea is representing the constituency tree as a sequence of split decisions that allows the usage of seq2seq framework to learn and decode the structure with high accuracy and faster speed.

[Chapter 5](#) investigates a new RST parsing approach. We propose a fast and effective discourse parser that allows parsing from scratch without requiring discourse segmentation as a prerequisite. The key idea is extending the parsing methods in [Chapter 4](#) to an unified discourse parsing model that could solve the discourse parsing and discourse segmentation in a single neural model.

[Chapter 6](#) introduces the strategies for auto learning the shallow hierarchical structure (spans) and incorporating local signals into attention-based neural network . The key idea is using pointing network to define the window that we want to focus on.

[Chapter 7](#) concludes the thesis and discusses several potential directions for future work.

Chapter 2

Literature Review

In this chapter, we review studies that are related to our work. First, we briefly describe widely used sequential neural networks which include recurrent neural networks and neural attention networks. Next, we present the pointing mechanism which helps select an element out of many elements of the input. We also provide brief descriptions of constituency and discourse parsing methods and then discuss the advantages and disadvantages of some popular parsers.

2.1 Neural Networks for Sequences

In this section, we review the two most popular neural networks for sequences in NLU: Recurrent and Attention-based Neural Networks.

2.1.1 Recurrent Neural Networks

Recurrent Neural Networks or RNNs [38] belong to a class of neural networks that can process sequential data of variable lengths. This section focuses on Long Short-Term Memory (LSTM) Networks [39], which is the most popular encoding unit for NLU.

Long Short-Term Memory LSTM is a powerful recurrent neural network specially designed to overcome the vanishing/exploding gradient issues in learning long-term dependencies. A vanilla LSTM unit consists of a memory cell, output state, an input gate, an output gate and a forget gate. Through three gates, the unit uses the memory cell and

output state to compute long-term and short-term values over arbitrary time intervals and control the flow of information. More specifically, the internal components of the LSTM are defined as follows:

$$\mathbf{f}_t = \sigma(\mathbf{W}_f \mathbf{x}_t + \mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{b}_f) \quad \text{forget gate} \quad (2.1)$$

$$\mathbf{i}_t = \sigma(\mathbf{W}_i \mathbf{x}_t + \mathbf{U}_i \mathbf{h}_{t-1} + \mathbf{b}_i) \quad \text{input gate} \quad (2.2)$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_o \mathbf{x}_t + \mathbf{U}_o \mathbf{h}_{t-1} + \mathbf{b}_o) \quad \text{output gate} \quad (2.3)$$

$$\mathbf{c}_t = \mathbf{f}_t \mathbf{c}_t + \mathbf{i}_t \tanh(\mathbf{W}_c \mathbf{x}_t + \mathbf{U}_c \mathbf{h}_{t-1} + \mathbf{b}_c) \quad \text{cell state} \quad (2.4)$$

$$\mathbf{h}_t = \mathbf{o}_t \tanh(\mathbf{c}_t) \quad \text{output state} \quad (2.5)$$

where σ and $\tanh$ are the sigmoid and tanh activation functions, respectively, and $\mathbf{i}_t, \mathbf{f}_t, \mathbf{o}_t$ indicate the input, forget and output gates respectively; $\mathbf{U}_*, \mathbf{W}_*, \mathbf{b}_*$ are the learnable parameters of the unit; $\mathbf{x}_t, \mathbf{h}_t, \mathbf{c}_t$ are the input state, hidden state and cell state at time step t . See Fig. 2.1 for an illustration. Noting that both $\mathbf{h}_t$ and $\mathbf{c}_t$ are also used as the input to the following unit at step $t + 1$. This enables the unit to remember its long-term and short-term memories at the same time.

FIGURE 2.1: An example of Long Short-Term Memory Network. Taken from [1].

Encoding/Decoding Application To encode sequences, RNN is often applied in bi-directional, where the input sequence is encoded in forward and backward directions. By concatenating the hidden representations of the two directions, the output layer can acquire information about the past and the future states, respectively, simultaneously. For decoding, RNNs could be used to generate the output sequence auto-regressively, i.e. decoding one token at a time, thanks to its ability to take into consideration past information for future predictions.

Another popular RNN variant is Gated Recurrent Unit (GRU) [40]. GRU can also be considered as a less-parameter variation of LSTM because it only keeps track of one hidden state and relies on two gates: update gate and reset gate. Under proper settings, GRU can obtain comparable performance while being faster to compute than LSTM [41].

2.1.2 Neural Attention Networks

The *attention mechanism* plays important roles in recent deep learning research innovation. The key defining characteristics of attention mechanism is the ability to dynamically select relevant context from a list of key representations with respect to a query representation. This concept is simple yet is proved to be very effective and powerful [2, 3]. This section covers several popular versions of attention neural networks.

Vanilla Attention This is the general form of attention mechanism in which the query vector attends to the sequence of key representations:

$$\mathbf{y} = \text{softmax}(\text{score}(\mathbf{q}, \mathbf{K}))\mathbf{V} \quad (2.6)$$

where $\text{softmax}(\mathbf{A})$ denotes the *softmax* operation over each row of matrix $\mathbf{A}$, $\mathbf{q} \in \mathbb{R}^d$ is the query vector, and $\mathbf{K}, \mathbf{V} \in \mathbb{R}^{n \times d}$ are the matrices containing the n key and value vectors respectively, with d being the number of vector dimensions. The output vector $\mathbf{y} \in \mathbb{R}^d$ can be interpreted as a weighted sum of the representations of the input value matrix $\mathbf{V}$. $\text{score}(\mathbf{q}, \mathbf{K})$ is a parametrized function that takes the input $\mathbf{q}$ and $\mathbf{K}$ to compute the score $\in \mathbb{R}^n$. Fig. 2.2 demonstrates how the output of attention network is computed.

Across the comprehensive history of attention mechanism, there are many scoring functions that are proposed. For example, Luong et al. [2] use tanh and Local Attention, Shaw et al. [42] proposed Relative Position Attention and Tay et al. [43] suggested Hermitian Attention to capture different properties of the sequence. However, the most popular scoring function is the dot-product attention due to its GPU-efficient computation.

FIGURE 2.2: An example of computing output of vanilla attention. Taken from [1].

Cross Attention Attention mechanism at first was proposed as an alignment mechanism to align target tokens to source tokens in machine translation. For this case, the key sequence and value sequence are computed from the source states while the query vector comes from the target side vectors:

$$\mathbf{y} = \text{softmax}(\text{score}(\mathbf{d}, \mathbf{E}_1)) \mathbf{E}_2 \quad (2.7)$$

where $\mathbf{E}_1, \mathbf{E}_2$ are neural transformations from the encoder states and $\mathbf{d}$ is computed from the decoder state. Generally, cross attention is used when we want to consider the alignment between two or more separated sequences. Fig. 2.3 illustrates application of cross attention in English to French translation. Specifically, at first step decoding phase, the decoder state $\mathbf{d}$ is computed from top-layer hidden state of the default input $\langle s \rangle$. Then the corresponding context vector is obtained by applying cross attention from $\mathbf{d}$ to encoder states. Next, the decoder state and context vector are combined to produce attention vector. Finally, the attention vector is used to make predictions.

FIGURE 2.3: An example of cross attention for Machine Translation. Taken from [2].

Self Attention This is a special case of attention where all the queries, keys and values are computed from the hidden representations of the same input sequence:

$$Y = \text{softmax}\left(\frac{(HW^Q)(HW^K)^T}{\sqrt{d}}\right)HW^V \quad (2.8)$$

where $W^Q, W^K, W^V \in \mathbb{R}^{d \times d}$ are the associated weights to perform linear transformations. Dot-product Attention is normalized by $\sqrt{d}$ in order to ensure the model stability. Fig. 2.4 shows self attentions applied to slight different context. In particular, “it” clearly refers to “animal” in the first sentence and refers to “street” in the second sentence. It is consistent with the attention weights in the model.

 FIGURE 2.4: An example of encoder self attention for the word “it” in different context. Taken from <https://bit.ly/2kDol2S>.

Attention Mask An attention mask is a boolean vector/matrix that decides the tokens of the key sequence that a query wants to attend to. Unlike the RNNs that strictly encode/decode the sequence in a sequential manner, the attention mask enables dynamic selection of the region to which we want to apply the attention. By choosing different forms of attention masks, the attention-based models can learn to recognize different patterns in the textual data. For example, masked language models [44, 45] randomly mask out tokens in the sequence and then try to predict them given the surrounding context. These models outperform their counterpart in recurrent networks in a number of downstream tasks.

FIGURE 2.5: Architecture of Transformer Network. Taken from [3].

Transformer Network Since its introduction in 2017, the Transformer [3] is the most popular choice of architecture for processing sequential data in sequence transduction problems (e.g., machine translation, speech recognition, text-to-speech transformation). It uses attention mechanisms in the encoder as well as decoder, therefore, removes the

need of recurrent neural network in sequence representation. With advantage of parallel computing in training and decoding; together with the ability to capture long-range dependency, Transformer and its variants have been a popular choice for deep learning tasks not only in natural language processing but also recently in computer vision.

We briefly describe the overall architecture of Transformer in Fig. 2.5. It consists of two components: an encoder and a decoder. The Transformer encoder is a stack of multiple identical layers, each of which has two sub-layers. The first is a multi-head attention mechanism and the second is a position-wise fully connected feed-forward network, both of which are followed by a residual connection [46] and a layer normalization [47]. In the encoder self attention, queries, keys and values are all from the output of previous encoder layer. The Transformer decoder also consists of a stack of multiple identical layers with similar sub-layers: masked multi-head attention mechanism (decoder self attention) and feed-forward network. Between these two sub-layers, the decoder inserts the third sub-layer of decoder cross attention. In the decoder cross attention, queries are from the output of previous decoder layer while keys and values are from the output of the encoder. The attention mask in the decoder allows the auto-regressive property of decoding procedure.

2.2 Pointing Mechanism

Typically, in a sequence-to-sequence (seq2seq) model, we have to define a fixed output vocabulary. The seq2seq framework encodes the input sequence into a sequence of vector representations and decodes/generates the output states step-by-step. At each decoding step, given the input sequence and output tokens that have already been generated, the network will produce vocabulary-size predictions for the next token. The next token is sampled from these predictions (*e.g.*, arg max, beam-search). While being effective in a lot of machine learning problems, this framework is not directly applicable for combinatorial problems where the output sequence consists of discrete tokens corresponding to positions in the input, in other words, output vocabulary is not fixed and depends on length of the input sequence. The pointing mechanism is proposed to crack these problems.

Vanilla Pointing Mechanism This technique at first was proposed in the seq2seq framework. It also computes attention weights from the decoder state to the encoder states. However, instead of computing context vector like attention mechanism, attention weights are directly used as the output distribution over the token of input. The pointing score is computed as follows:

$$\mathbf{u}_i = \mathbf{V} \tanh(\mathbf{E}\mathbf{W}_1 + \mathbf{d}_i\mathbf{W}_2) \quad (2.9)$$

$$\mathbf{a}_i = \text{softmax}(\mathbf{u}_i) \quad (2.10)$$

where $\mathbf{E} = (\mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_n) \in \mathbb{R}^{n \times d}$ is the matrix containing the n encoder states and $\mathbf{d}_i \in \mathbb{R}^d$ is the decoder state at time step i . Fig. 2.6 illustrates application of pointing mechanism in a convex hull problem with four points. Fig. 2.6(a) shows how to formulate the convex hull problem in a seq2seq framework. The input sequence consists of point coordinates and the output sequence includes the point indices that form the convex hull. In Fig. 2.6(b), blue and purple boxes represent encoder states and decoder states, respectively. From the last encoder states and start token “=>”, decoder computes the first decoder state and then modulates a pointing mechanism over input sequence. The resulting pointing score is used to make predictions and the coordinates of the predicted point fed to the next decoding step. This process is repeated until the predicted point is the end token “<=”.

Pointing mechanism can solve other combinatorial problems such as finding Delaunay triangle, travelling salesman problems, segmentaion and parsing [4]. In NLP, we can use it for structured prediction problem like dependency parsing [48] and discourse parsing [49]. In this thesis, we use this mechanism to build conditional constituency pointing parser (Chapter 4) and end-to-end RST parser (Chapter 5). This is really useful and there has been a lot of work done on making it easy and fast to serve deep neural network predictions.

Generalized Pointing Mechanism Similar to attention mechanism, the pointing mechanism can be generalized for any arbitrary query vector and key matrices:

$$\mathbf{u}_q = \mathbf{q}\mathbf{W}^Q(\mathbf{K}\mathbf{W}^K)^T \quad (2.11)$$

FIGURE 2.6: Vanilla pointing network with finding convex hull problem. Taken from [4].

$$\mathbf{a}_q = \text{softmax}(\mathbf{u}_q) \quad (2.12)$$

with the pointing is from a query $\mathbf{q} \in \mathbb{R}^d$ to sequence of keys additive $\mathbf{K} \in \mathbb{R}^{n \times d}$. This version does not limit pointing mechanism to encoder-decoder framework and also use dot product attention which is faster and more efficient than other attention mechanisms. In this thesis, we use this mechanism to build span-based constituency pointing parser (Chapter 3) and derive the dynamic window attention (Chapter 6).

2.3 Constituency Parsing

Constituency structure is the idea that consecutive words can be combined into nested phrases, spans or constituents. Constituency parsing is the process of organizing a sentence into nested constituents based on the grammar of the language. It is an essential process that humans instinctively do to understand what others say and to deliver their messages. For instance, when a person hears “She enjoys playing tennis.”, he or she would subconsciously figure the underlying tree structure like the parse tree shown in Fig. 2.7. Humans understand the noun “she” is about a woman; “enjoys” is a verb; verb phrase “playing tennis” indicates that “she” is interested in the hobby “tennis”. Even if it looks easy for humans to parse sentences with high accuracy, it is not that easy for the machines

to do so. One of the most serious challenges for constituency parser is the structural ambiguity where a sentence could be parsed to more than one valid constituency tree. Moreover, the sentence's parse tree search space is exponential with regard to sentence length n , hence developing an efficient and effective parsing method is difficult.

Definition We can define the constituency parsing as the problem of finding a set of labeled spans over the input sentence. Let $\mathcal{S}(T)$ denote the set of *labeled spans* for a parse tree T . Formally, $\mathcal{S}(T)$ can be expressed as

$$\mathcal{S}(T) := \{((i_t, j_t), l_t)\}_{t=1}^{|\mathcal{S}(T)|} \text{ for } i_t < j_t \quad (2.13)$$

where $|\mathcal{S}(T)|$ is the number of spans in the tree.

In order to build AI that people can communicate and collaborate to, it is important to build a human level parser. Many NLU applications use sentences' parse trees as their inputs to give the models a more suitable inductive bias and more structural information about the data. Leveraging the structural prior is beneficial, as showed in machine translation and text classification [50–52]. Therefore, devising effective and efficient algorithms for parsing has been a key focus in NLP.

FIGURE 2.7: A parse tree for “She enjoys playing tennis.”

With the advancements in neural approaches, various neural architectures have been proposed for constituency parsing as they are able to effectively encode the input tokens

into dense contextual vector representations while modeling the structural dependencies between tokens in a sentence. These include recurrent networks [21, 53] and more recently self-attentive networks [27]. The parsing methods can be distinguished based on how they deal with the labeled structure training and prediction. The most popular approaches are the chart-based and transition-based models. In this section, first we briefly introduce these two kinds of parsers, and then present some other approaches.

2.3.1 Chart-based Parsers

Before the rising of neural network, chart-based parsers employed statistical models that make use of probabilistic context free grammar (PCFG) rules to guide the tree generation procedure. Each PCFG rule expresses the way a syntactic label can be split to one or more syntactic labels with some probabilities. For example, a noun phrase (NP) can be split into a ProperNoun or a determiner (Det) followed by a Nominal. Each of these rules has a corresponding probability (e.g., $P(NP \rightarrow \text{Proper-Noun}) = 0.2$, $P(NP \rightarrow \text{Det Nomial}) = 0.4$). Most of the earlier work focus on enriching PCFG rules with head lexicalization, fine-grained structural annotation or automatic latent variables [54–57]. There are attempts to apply conditional random fields (CRF) to take into consideration of future decisions/features on these parsing methods [58].

With the advancement of neural networks, the context free grammar can be modelled in different ways. Instead of learning probability for each PCFG rule, these neural parsers train a scorer network to assign a score to each eligible span of the sentence. Fig. 2.8 illustrates an example of span scoring. The figure lays out an example representation for the span (1, 3), which corresponds to the noun phrase “the flight”. As we can see, word representations are obtained by passing the input word through an encoder network like RNNs or Transformer. Then, the representation of a span is computed by taking the difference between representations of its start and end tokens. A multi-layer perceptron (MLP) then is applied to the resulting span representation to produce a span score. There are many ways to choose the scoring function: Gaddy et al. [26], Kitaev and Klein [27], Stern et al. [59] only use a single score function modeling jointly span and label, while

Zhang et al. [60] use a scalar-score function and a multiclass-score function for structure building and labeling respectively.

FIGURE 2.8: An example of computing a span score. Taken from [5].

A parsing inference algorithm (*e.g.*, greedy, CKY) is then applied to combine the span scores and construct the best-scoring tree. The span combination mechanism can be considered as an alternative way to implicitly learn the PCFG rules. To be effective, these span-based parsers need to train by margin-based algorithm which requires structured inference during training. Despite getting high accuracy, the greedy and CKY methods need at least $\mathcal{O}(n^2)$ and $\mathcal{O}(n^3)$ running steps to infer the tree structure, respectively.

2.3.2 Transition-based Parsers

The main idea of transition-based parsers is to cast the tree structure into a sequence with transition states and actions, where states denote the partial tree, and actions specify next steps state-transition operations. By the transition actions, the constituency tree is constructed incrementally. The transition actions, typically include the *SHIFT* operation to build terminal nodes, the *REDUCE* operation to complete the subtrees or terminal nodes

and $NT(X)$ operation to assign label X to the subtrees or nodes. Fig. 2.9 shows an example of how to obtain a transition sequence from a constituency tree. We traverse in the depth-first order to build the tree: at each stage, processing a node, processing a word or closing a node correspond to the $NT(X)$, $SHIFT$, $REDUCE$ operations, respectively.

Input: *The hungry cat meows .*

	Stack	Buffer	Action
0		<i>The hungry cat meows .</i>	NT(S)
1	(S	<i>The hungry cat meows .</i>	NT(NP)
2	(S (NP	<i>The hungry cat meows .</i>	SHIFT
3	(S (NP <i>The</i>	<i>hungry cat meows .</i>	SHIFT
4	(S (NP <i>The hungry</i>	<i>cat meows .</i>	SHIFT
5	(S (NP <i>The hungry cat</i>	<i>meows .</i>	REDUCE
6	(S (NP <i>The hungry cat</i>)	<i>meows .</i>	NT(VP)
7	(S (NP <i>The hungry cat</i>) (VP	<i>meows .</i>	SHIFT
8	(S (NP <i>The hungry cat</i>) (VP <i>meows</i>	<i>.</i>	REDUCE
9	(S (NP <i>The hungry cat</i>) (VP <i>meows</i>)	<i>.</i>	SHIFT
10	(S (NP <i>The hungry cat</i>) (VP <i>meows</i>) .		REDUCE
11	(S (NP <i>The hungry cat</i>) (VP <i>meows</i>) .)		

FIGURE 2.9: An example of the transition sequence of a parse tree. Taken from [6].

By casting constituency tree as a sequence of transition operations, transition-based parsers can apply sequence generation mechanism for constituency parsing. These methods are generally faster than the chart-based methods. However, most transition-based parsers perform greedy searching for the best resulting constituent tree, which may be vulnerable to the error propagation problem since the current step errors can affect the future predictions. Several studies [22, 61] propose dynamic oracles to overcome this issue. However, one of the disadvantages of these methods is that parts of the training data with inaccurate transition operations and their oracle operations need to be generated and added to training data. This augmented methods may cost extra efforts without much in accuracy.

2.3.3 Other Parsers

With the advancement of neural networks, researchers also try to cast the parsing problem into tasks that can be solved differently. One representative attempt is to treat the tree as a regression problem. For example, Gómez and Vilares [62] and Shen et al. [63] proposed

to map the syntactic tree of a sentence containing n tokens into a sequence of $n - 1$ scalar labels. However, parsers of this type suffer from the exposure bias during inference. Beside these methods, neural sequence-to-sequence models have been used to generate a linearized form of the tree [64–67]. However, these methods may generate invalid trees when the open and end brackets do not match. In addition, several studies attempt to integrate external information to benefit the parsing tasks. For example, Zhou and Zhao [68] uses external supervision (*head* information) from the dependency parsing to achieve better performance.

2.3.4 Evaluation Metrics

FIGURE 2.10: An example of a predicted tree and its gold tree.

Figures 2.10 illustrates an example of a predicted tree and its gold tree. We measure parsing accuracy with respect to the constituent in the hypothesis parse tree. It reflects the ability of our model to correctly identify labeled constituents in a hand-labeled, reference parse tree. Let g be the total number of labeled constituents in the human annotation, h be the total number of labeled constituents in the model prediction, and c be the total number of correct labeled constituents in the model prediction. Then, we measure Precision, Recall, and F-score for parser performance as follows:

$$\text{labeled precision} = \frac{c}{h} \quad (2.14)$$

$$\text{labeled recall} = \frac{c}{g} \quad (2.15)$$

$$\text{labeled F-score} = \frac{2c}{g + h} \quad (2.16)$$

Figure 2.11 illustrates an example of computing the evaluation metrics with a sentence. In the reference tree, there are five spans in total. On the other hand, predicted tree has three spans and they are all correct. According to the above equations, we can obtain the scores as follows:

Gold Spans	Predicted Spans	Match
$NP(w_{1:1})$	$NP(w_{1:1})$	✓
$NP(w_{4:4})$	-	X
$S - VP(w_{3:4})$	-	X
$VP(w_{2:4})$	$VP(w_{2:4})$	✓
$S(w_{1:5})$	$S(w_{1:5})$	✓
Scores	$Precision = 3/3, Recall = 3/5, F1 = 3/4$	

FIGURE 2.11: An example of an evaluation on constituency parsing

2.4 Discourse Parsing

A text is not built from independent textual units but of related and structured units. In fact, each clause or sentence in a text are connected to the nearby clauses or sentences logically, which form a coherence whole to be interpreted clearly. Discourse analysis, which focuses on the underlying coherence structure of the text, is an important research area in NLP. It could contribute to myriad number of downstream NLP applications such as text summarization and compression, text generation, machine translation, sentiment analysis, information extraction, and question answering.

Early work has developed many discourse theories to describe coherence structures from different points of view. The most popular ones consist of lexical cohesion [11], intentional structure theory [12], RST [13], centering theory [14], discourse representation theory (DRT) [15], and discourse-lexicalized tree adjoining grammar (DLTAG) theory [16]. Among them, RST is probably the most commonly used theory. It represents a text by labeled hierarchical structures called discourse trees where the node labels indicate the

relation between two child spans of text. Unlike constituency tree, the atomic units in a discourse tree are text clauses, *a.k.a.* Elementary Discourse Units (EDUs). The relationship represents not only the coherence relation between two adjacent spans of text but their relative importance in the text: nuclei and satellites are the relation's the main and the supportive parts, respectively. For example, in Fig. 2.12, there are six EDUs in our text. *Same-Unit* represents the relation between the clauses "But the things it's supposed to measure -- manufacturing strength --" and "it missed altogether last month.". In this example, the two clauses are at the same level of importance, which means each clause is labeled as *nuclei* with regard to the other.

FIGURE 2.12: Example of a discourse tree. Taken from [7].

Definition Assuming that a document has already been segmented into EDUs, following the traditional approach, the corresponding discourse tree (DT) can be represented as a set of labeled constituents.

$$\mathbb{C} := \{((i_t, k_t, j_t), r_t) | i_t \leq k_t < j_t\}_{t=1}^m \quad (2.17)$$

where $m = |\mathbb{C}|$ is the number of internal nodes in the tree and r_t is the relation label between the discourse unit containing EDUs i_t through k_t and the one containing EDUs $k_t + 1$ through j_t .

One challenge of discourse parsing is the lack of large scale annotated data due to the expensive and time-consuming annotated process. Another challenge is extracting effective features to capture the complicated context. Unlike constituency parsing where we deal with sentence, discourse parsing normally involves document-level parsing. Dealing

with raw and arbitrary long texts need special approaches to capture the relevant context information. Last but not least, finding discourse structure generally requires breaking the text into EDUs (discourse segmentation) first and linking the EDUs into a DT (discourse parsing) later. Usually, the discourse parser assumes that the input text has already been segmented into EDUs in a prerequisite step. Therefore, the errors in segmentation (if any) would affect the overall parsing performance [7, 69].

Before the explosion of deep learning, researchers used various linguistically motivated features to improve the performance. Thanks to powerful feature representation learning of deep learning, researchers can avoid the feature engineering which requires domain expertise and hardcore feature extraction.

Discourse structures in RST share the similar structure as the constituency syntax tree (section 2.3), therefore, many tree reconstruction algorithms are inspired by constituency parsing. However, there are two main differences between two parsing tasks. The first one is that the fundamental units in a discourse tree are EDUs rather than words in a constituency tree. Therefore, it is important to model the EDU representations effectively in discourse parsing. Second difference is that the node labels of the discourse tree reflect the relationship between two or more child nodes. This indicates that labeling tasks are done after structure building. This is not common in constituency parsing because the constituency parsers often model the structure and label jointly. These differences affect how the discourse parser learn and make predictions. Similar to constituency parsing, first we introduce chart-based and transition-based parsers, and then present some other approaches.

2.4.1 Chart-based Parsers

For chart-based discourse parsing, there are two separated neural score functions for structure and label prediction. The structure classifier is a binary classifier to decide whether two input clauses can be merged into a larger clauses. The labeling classifier is a multi-class classifier to predict the coherence relation between the two child clauses. The score for a discourse constituent is obtained by combining these two classifiers. Fig. 2.13 shows an example of these parsers [8, 70]: a parent span representation is computed recursively

FIGURE 2.13: An example of a chart-based discourse parser. Taken from [8].

based on its child representations. At each merging step, the distributed vectors for the merged point are calculated by passing the concatenation of child span representations through the corresponding classifier networks.

Given the score for all possible discourse spans, the CKY-like parsing algorithm decodes the highest scoring tree through dynamic programming. While these methods are superior to the greedy parsers, the parser is quite slow with the time complexity of at least $\mathcal{O}(n^3)$. Moreover, these methods also need to infer the structure in training, which may not be efficient.

2.4.2 Transition-based Parsers

Stack	Buffer	Prediction
ϵ	ϵ	GEN(e_1)
EDU(e_1)	e_1	GEN(e_2)
EDU(e_1) EDU(e_2)	$e_1 e_2$	GEN(e_3)
EDU(e_1) EDU(e_2) EDU(e_3)	$e_1 e_2 e_3$	RE(ATTR, SN)
EDU(e_1) (Unit(ATTR, SN) EDU(e_2) EDU(e_3))	$e_1 e_2 e_3$	RE(JUST, NS)
(Unit(JUST, NS) EDU(e_1) (Unit(ATTR, SN) EDU(e_2) EDU(e_3)))	$e_1 e_2 e_3$	

[e_1 Acme Inc. has closed several widget factories.] [e_2 The CEO told investors] [e_3 they were no longer profitable.]

FIGURE 2.14: An example of a discourse tree's transition operation sequence (source[9]).

Similar to constituent parsing, a transition-based discourse parser uses a sequence of shift-reduce operation decisions to construct the discourse tree incrementally. Fig. 2.14 illustrates an example of casting a discourse tree into a transition sequence in a bottom-up manner. The parsing algorithm can be bottom-up [71–73] or top-down [49]. At each time step, the models determine the locally best next operation (*merge* or *split*) based on the current parsing status (partial tree). They use shift-reduce parsing methods to first construct a tree structure, and then identify the relation and nuclearity labels. The algorithm’s time complexity is linear with respect to the number of EDUs. The parsers in this category are greedy which is vulnerable to the exposure bias problem. Moreover, the smallest unit in these parsers is EDU. Therefore, the errors in discourse segmentation would affect the end-to-end performance significantly.

2.4.3 Other Parsers

There are several attempts to solve discourse parsing in other ways that do not fall in the above categories. One strategy is to employ greedy two-stage bottom-up parser. This idea is first proposed by HILDA (high-level discourse analyzer) [74]. It involves a binary structure classification model and a multi-label classifier for structure and label predictions, respectively. At first, the binary classifier finds two most relevant consecutive units to concatenate, then the multi-label classifier determines the discourse relation between the two units. This process iterates until all units are merged into a tree. This algorithm’s running complexity is $\mathcal{O}(n^2)$ in term of the number of EDUs. While being simple and efficient, this strategy faces the issues include its error propagation problem and the difficulty of taking context into account.

Another strategy is to build the discourse tree bottom-up at two stages: intra-sentential and multi-sentential [7, 75–77]. The first stage performs intra-sentential parsing in order to build the discourse tree for the individual sentences. The second stage connect these discourse trees and build the final structure for the whole document. The two separate parsing models could utilize the fact that intra and multi-sentential parsing could have their own feature sets and this separation can deal with the scalability issue of huge number of EDUs

in document. However, this parsing method would suffer from “leaky” boundaries issues, where discourse structures violate sentence boundaries [7].

2.4.4 Evaluation Metrics

	Predicted Spans	Correct	Nuclearity	Correct	Relation	Correct
Parseval	π_2, π_3	X	NN	X	-	X
	π_1, π_3	✓	NS	X	R2	X
	π_1, π_4	✓	NS	✓	R1	✓
	Scores	$S = 2/3$		$N = 1/3$	$R = 1/3$	$F = 1/3$
RST-Parseval	π_1	✓	N	✓	SPAN	✓
	π_2	✓	N	X	R3	X
	π_3	✓	N	X	R3	✓
	π_4	✓	S	✓	R1	✓
	π_2, π_3	X	S	X	R2	✓
	π_1, π_3	✓	N	✓	SPAN	✓
	π_1, π_4	✓	R	✓	ϵ	✓
	Scores	$S = 6/7$		$N = 5/7$	$R = 5/7$	$F = 5/7$

FIGURE 2.15: An example of a constituent-based evaluation on discourse parsing. Taken from [10]

The evaluation procedures for RST discourse parsers are inspired by the measurement for evaluating syntactic parsing [78]. The basic metrics include constituent precision, recall and F_1 measures. To report the accuracy of a discourse parser, micro-average F1 scores

are computed and reported mostly in four aspects: the correctness of spans (S), nuclearity (N), relations (R) and Full (F) (both N and R) of constituents. Figure 2.15 illustrates two most popular evaluation metrics: Parseval [10] and RST-Parseval [78]. The main difference between two metrics is how they distribute the labels in the tree to evaluate. In the Parseval, labels are kept in the parent nodes so the single EDU evaluation is ignored. In the RST-Parseval, labels are split to child nodes so the single EDU evaluation is still considered. Normally, the scores in the Parseval are less than the scores in the RST-Parseval.

Chapter 3

Efficient Constituency Parsing by Pointing

In this chapter, we propose a novel constituency parsing model that casts the parsing problem into a series of pointing tasks. Specifically, our model estimates the likelihood of a span being a legitimate tree constituent via the pointing score corresponding to the boundary words of the span. Our parsing model supports efficient top-down decoding and our learning objective is able to enforce structural consistency without resorting to the expensive CKY inference. The experiments on the standard English Penn Treebank parsing task show that our method achieves 92.78 F1 without using pre-trained models, which is higher than all the existing methods with similar time complexity. Using pre-trained BERT, our model achieves 95.48 F1, which is competitive with the state-of-the-art while being faster. Our approach also establishes new state-of-the-art in Basque and Swedish in the SPMRL shared tasks on multilingual constituency parsing.¹

3.1 Introduction

Constituency or phrase structure parsing is a core task in natural language processing (NLP) with myriad downstream applications. Therefore, devising effective and efficient algorithms for parsing has been a key focus in NLP.

¹This chapter is published as *Efficient Constituency Parsing by Pointing*, Proceedings of **ACL 2020** [79]

With the advancements in neural approaches, various neural architectures have been proposed for constituency parsing as they are able to effectively encode the input tokens into dense vector representations while modeling the structural dependencies between tokens in a sentence. These include recurrent networks [21, 53] and more recently self-attentive networks [27].

As described in Chapter 2, the parsing methods can be broadly distinguished based on whether they employ a greedy transition-based algorithm or a globally optimized chart parsing algorithm. The transition-based parsers [21–23] generate trees autoregressively as a form of shift-reduce decisions. Though computationally attractive, the local decisions made at each step may propagate errors to subsequent steps which would suffer from exposure bias.

Chart parsing methods, on the other hand, learn scoring functions for subtrees and perform global search over all possible trees to find the most probable tree for a sentence [25–28]. In this way, these methods can ensure consistency in predicting structured output. The limitation, however, is that they run slowly at $\mathcal{O}(n^3)$ or higher time complexity.

In this chapter, we propose a novel parsing approach that casts constituency parsing into a series of pointing problems (Figure 3.1). Specifically, our parsing model estimates the pointing score from one word to another in the input sentence, which represents the likelihood of the span covering those words being a legitimate phrase structure (*i.e.*, a subtree in the constituency tree). During training, the likelihoods of legitimate spans are maximized using the cross entropy loss. This enables our model to enforce structural consistency, while avoiding the use of structured loss that requires expensive $\mathcal{O}(n^3)$ CKY inference [26, 27]. The training in our model can be fully parallelized without requiring structured inference as in [62, 63]. Our pointing mechanism also allows efficient top-down decoding with a best and worse case running time of $\mathcal{O}(n \log n)$ and $\mathcal{O}(n^2)$, respectively.

In the experiments with English Penn Treebank parsing, our model without any pre-training achieves 92.78 F1, outperforming all existing methods with similar time complexity. With pre-trained BERT [80], our model pushes the F1 score to 95.48, which is on par with the state-of-the-art [28], while supporting faster decoding. Our model also performs competitively on the multilingual parsing tasks in the SPMRL 2013/2014 shared tasks and establishes new state-of-the-art in Basque and Swedish.

3.2 Related Work

Prior to the neural tsunami in NLP, parsing methods typically model correlations in the *output space* through probabilistic context-free grammars (PCFGs) on top of sparse (and discrete) input representations either in a generative regime [56] or a discriminative regime [81] or a combination of both [82]. Beside the chart parser approach, there is also a long tradition of transition-based parsers [83]

Recently, however, with the advent of powerful neural encoders such as LSTMs [39], the focus has been switched more towards effective modeling of correlations in the *input's latent space*, as the output structures are nothing but a function of the input [26]. Various neural network models have been proposed to effectively encode the dense input representations and correlations, and have achieved state-of-the-art parsing results. To enforce the structural consistency, existing neural parsing methods either employ a transition-based algorithm [21, 23, 84] or a globally optimized chart-parsing algorithm [26, 27].

Meanwhile, researchers also attempt to convert the constituency parsing problem into tasks that can be solved in alternative ways. For instance, Fernández-González and Martins [85] transform the phrase structure into a special form of dependency structure. Such a dependency structure, however, requires certain corrections while converting back to the corresponding constituency tree. Gómez and Vilares [62] and Shen et al. [63] propose to map the constituency tree for a sentence of n tokens into a sequence of $n - 1$ labels or scalars based on the depth or height of the lowest common ancestors between pairs of consecutive tokens. In addition, methods like [86, 87] apply the sequence-to-sequence framework to “translate” a sentence into the linearized form of its constituency tree. While being trivial and simple, parsers of this type do not guarantee structural correctness, because the syntax of the linearized form is not constrained during tree decoding.

Our approach in this chapter differs from previous work in that it represents the constituency structure as a series of span-based pointing representations and has a relatively simpler cross entropy based learning objective. The pointing representations can be computed in parallel, and can be efficiently converted into a full constituency tree using a top-down algorithm. Our pointing mechanism shares certain similarities with the Pointer

Network [4], but is distinct from it in that our method points a word to another word within the same encoded sequence.

3.3 Proposed Models

Following the standard practice in parsing [26, 63], we convert the n -ary tree into a right-branching binary form. The new created nodes, which are not labelled in the original tree, are labeled using a dummy label $\emptyset$. Similarly, the labels in unary chains corresponding to nested labelled spans are collapsed into unique atomic labels, such as S-VP in Fig. 3.1.

Although our method shares the same “span-based” view with that of Stern et al. [59], our approach diverges significantly from their framework in the way we treat the whole parsing problem, and the representation and modeling of the spans, as we describe below.

3.3.1 Parsing as Pointing

In contrast to previous approaches, we cast parsing as a series of pointing decisions. For each index i in the input sequence, the parsing model points it to another index p_i in order to identify the tree span (i, p_i) , where $i \neq p_i$. Similar to Pointer Networks [4], each pointing mechanism is modeled as a multinomial distribution over the indices of the input tokens (or encoder states). However, unlike the original pointer network where a decoder state points to an encoder state, in our approach, every encoder state $\mathbf{h}_i$ points to another encoder state $\mathbf{h}_{p_i}$.

In this chapter, we generally use $x \rightarrow y$ to mean x points to y . We will refer to the pointing operation either as a function of the encoder states (e.g., $\mathbf{h}_i \rightarrow \mathbf{h}_{p_i}$) or simply the corresponding indices (e.g., $i \rightarrow p_i$). They both mean the same operation where the pointing function takes the encoder state $\mathbf{h}_i$ as the query vector and points to $\mathbf{h}_{p_i}$ by computing an attention distribution over all the encoder states.

Let $\mathcal{P}(T)$ denote the set of pointing decisions derived from a tree T by a transformation $\mathcal{H}$, i.e., $\mathcal{H} : T \rightarrow \mathcal{P}(T)$. For the parsing process to be valid, the transformation $\mathcal{H}$ and its inverse $\mathcal{H}'$ which transforms $\mathcal{P}(T)$ back to T , should both have a one-to-one mapping property. Otherwise, the parsing model may confuse two different parse trees with the

FIGURE 3.1: A binarized constituency tree for the sentence “She enjoys playing tennis.”. The node $S-VP$ is an example of a collapsed atomic label. We omit POS tags and singleton spans for simplicity. Below the tree, we show span and pointing representations of the tree.

same pointing representation. In this chapter, we propose a novel transformation that satisfies this property, as defined by the following proposition.

Proposition 1. Given a binary constituency tree T for a sentence containing n tokens, the transformation $\mathcal{H}$ converts it into a set of pointing decisions $\mathcal{P}(T) = \{(i \rightarrow p_i, l_i) : i = 1, \dots, n-1; i \neq p_i\}$ such that $(\min(i, p_i), \max(i, p_i))$ is the **largest** span that starts or ends at i , and l_i is the label of the nonterminal associated with the span.

To elaborate further, each pointing decision in $\mathcal{P}(T)$ represents a specific span in $\mathcal{S}(T)$. The pointing $i \rightarrow p_i$ is directional, while the span that it represents (i', j') is non-directional. In other words, there may exist position i such that $i > p_i$, while $i' < j' \forall i', j' \in [1, n]$. In fact, it is easy to see that if the token at index i is a left-child of a subtree, the largest span involving i starts at i , and in this case $i < p_i$ and $i' = i, j' = p_i$. On the other hand, if the token is a right-child of a subtree, the respective largest span ends at position i , in which case $i > p_i$ and $i' = p_i, j' = i$ (e.g., see $4 \rightarrow 2$ in Figure 3.1). In addition, as the spans in $\mathcal{S}(T)$ are unique, it can be shown that the pointing decisions in $\mathcal{P}(T)$ are also distinct from one another.

Algorithm 1: Convert binary tree to Pointing

Require: Binary tree T and its span representation $\mathcal{S}(T)$
Ensure: Pointing representation $\mathcal{P}(T)$
 $\mathcal{P}(T) = []$ {Empty pointing list}
for each leaf_i in T **do**
 $\text{node} \leftarrow \text{leaf}_i$
 $(x, y) \leftarrow (i, i)$ {Initialize current span, $x \leq y$ }
 $l_i \leftarrow \emptyset$ {Initialize label of current span}
 while $x = i$ or $y = i$ **do**
 $p_i \leftarrow x + y - i$
 $l_i \leftarrow \text{node.label}$ {The span's label}
 $\text{node} \leftarrow \text{node.parent}$
 $(x, y) \leftarrow \text{node.span}$ {Span covered by node}
 end while {Until i is no longer start/end point}
 push($\mathcal{P}(T)$, $(i \rightarrow p_i, l_i)$)
end for
return $\mathcal{P}(T)$

Given such pointing formulation, for every constituency tree, there exists a trivial case $(1 \rightarrow n, l_1)$ where $p_1 = n$ and l_1 is generally ‘S’. Thus, to make our formulation more general with n inputs and n outputs and convenient for the method description discussed later on, we add another trivial case $(n \rightarrow 1, l_1)$. With this generalization, we can represent the pointing decisions of any binary constituency tree T as:

$$\mathcal{P}(T) = \{(i \rightarrow p_i, l_i) : i = 1, \dots, n; i \neq p_i\} \quad (3.1)$$

The pointing representation of the tree in Figure 3.1 is given at the bottom of the figure. To illustrate, in the parse tree, the largest phrase that starts or ends at token 2 (‘enjoys’) is the subtree rooted at ‘ $\emptyset$ ’, which spans from 2 to 5. In this case, the span *starts* at token 2. Similarly, the largest phrase that starts or ends at token 4 (‘tennis’) is the span “enjoys playing tennis”, which is rooted at ‘VP’. In this case, the span *ends* at token 4.

Algorithm 1 describes the procedure to convert a binary tree to its corresponding pointing representation. Specifically, from each leaf token i , the algorithm traverses upward along the hierarchy until the non-terminal node that does not start or end with i . In this way, the largest span starting or ending with i can be identified.

Proof of Proposition 1 Given $\mathcal{P}(T) = \{(i \rightarrow p_i, l_i) : i = 1, \dots, n-1; i \neq p_i\}$, generated from tree T (here we omit the unary leaves and POS-tags), we at first define the inverse $\mathcal{H}'$ as follows:

$$\mathcal{H}'(\mathcal{P}(T)) = \{((\min(i, p_i), \max(i, p_i)), l_i) : i = 1, \dots, n-1\}$$

We would prove $\mathcal{H}'(\mathcal{P}(T)) = T$.

A binary tree T has exactly $n-1$ internal nodes (or spans). It is noteworthy to mention that for each pointing $(i \rightarrow p_i, l_i)$, $((\min(i, p_i), \max(i, p_i)), l_i)$ is a span in T . As we consider i from 1 to $n-1$, there are totally at most $n-1$ such spans in $\mathcal{H}'(\mathcal{P}(T))$ (we do not know whether these spans are not be distinct). Therefore, if we can prove that all $((\min(i, p_i), \max(i, p_i)), l_i)$ spans are distinct for $i = 1, \dots, n-1$, $\mathcal{H}'(\mathcal{P}(T))$ will cover all the span in T , which means $\mathcal{H}'(\mathcal{P}(T)) = T$. We prove this by contradiction.

Assume that there exist $i, j \in \{1, \dots, n-1\}$ such that $(\min(i, p_i), \max(i, p_i)) = (\min(j, p_j), \max(j, p_j))$ for $j \neq i$. First, if $p_i = n$, according to the above condition, $(\min(j, p_j), \max(j, p_j)) = (\min(i, n), \max(i, n)) = (i, n)$. This means, either $j = n$ or $j = i$, which contradicts with our initial assumption that $j \neq i$ and $j \in \{1, \dots, n-1\}$. So, p_i cannot be equal to n . Similarly, we can prove that p_j also cannot be equal to n . Thus, we can conclude that $p_i, p_j \in \{1, \dots, n-1\}$. Now, without loss of generality, let us assume that $j > i$. With this assumption, the two spans will be identical if and only if $p_i = j$ and $p_j = i$. In this case, the span (i, j) would be the largest span that starts with i and ends at j . However, since $1 \leq i < j \leq n-1$, the span (i, j) must be a left or right child of another (parent) span. If (i, j) is the left child, its parent span needs to start with i , making it larger than (i, j) . This contradicts to the property that $(i, j) = (i, p_i)$ is the largest span that starts or ends at i . Similarly, if (i, j) is the right child, its parent span needs to end at j , making it larger than (i, j) . This again contradicts to the property that $(j, i) = (j, p_j)$ is the largest span that starts or ends at j .

In conclusion, we have $\mathcal{H}'(\mathcal{P}(T)) = T$. This will guarantee that $\mathcal{H}$ and $\mathcal{H}'$ are one-to-one: If there exist T_1, T_2 such that $\mathcal{P}(T_1) = \mathcal{P}(T_2)$, we would have $\mathcal{H}'(\mathcal{P}(T_1)) = \mathcal{H}'(\mathcal{P}(T_2))$ or $T_1 = T_2$. If there exist T_1, T_2 such that $\mathcal{H}'(\mathcal{P}(T_1)) = \mathcal{H}'(\mathcal{P}(T_2))$, we would have $T_1 = T_2$.

3.3.2 Top-Down Tree Inference

In the previous section, we described how to convert a constituency tree T into a sequence of pointing decisions $\mathcal{P}(T)$. We use this transformation to train the parsing model (described in detail in Sections 3.3.3 - 3.3.4). During inference, given a sentence to parse, our decoder with the help of the parsing model predicts $\mathcal{P}(T)$, from which we can construct the tree T . However, not all sets of pointings $\mathcal{P}(T)$ guarantee the generation of a valid tree. For example, for a sentence with four (4) tokens, the pointing $\mathcal{P}(T) = \{(1 \rightarrow 4, l_1), (2 \rightarrow 3, l_2), (3 \rightarrow 4, l_3), (4 \rightarrow 1, l_1)\}$ does not generate a valid tree because 3^{rd} token cannot belong to both spans $(2, 3)$ and $(3, 4)$. In other words, simply taking the $\arg \max$ over the pointing distributions may not generate a valid tree.

Our approach to decoding is inspired by the span-based approach of Stern et al. [59]. In particular, to reduce the search space, we score for span identification (given by the pointing function) and label assignment separately.

Span Identification. We adopt a top-down greedy approach formulated as follows.

$$k^* = \arg \max_k s_{\text{split}}(i, k, j) \quad (3.2)$$

where $s_{\text{split}}(i, k, j)$ is the score of having a split-point at position k ($i \leq k < j$), as defined by the following equation.

$$s_{\text{split}}(i, k, j) = \rho(k \rightarrow i) + \rho(k+1 \rightarrow j) \quad (3.3)$$

where $\rho(k \rightarrow i)$ and $\rho(k+1 \rightarrow j)$ are the pointing scores (probabilities) for spans (i, k) and $(k+1, j)$, respectively. Note that the pointing scores are *asymmetric*, meaning that $\rho(i \rightarrow j)$ may not be equal to $\rho(j \rightarrow i)$, because pointing from i to j is different from pointing from j to i . This is different from previous approaches, where the score of a span is defined to be symmetric. We build a tree for the input sentence by computing Eq. 3.2 recursively starting from the full sentence span $(1, n)$.

In the general case when $i < k < j - 1$, our pointing-based parsing model should learn to assign high scores to the two spans (i, k) and $(k+1, j)$, or equivalently the pointing

decisions $k \rightarrow i$ and $k+1 \rightarrow j$. However, the pointing formulation described so far omits the trivial *self-pointing* decisions, which represent the *singleton spans*. A singleton span is only created when the splitting decision splits an n -size span into a single-token span (singleton span) and a sub-span of size $n-1$, *i.e.*, when $k = i$ or $k = j-1$. For instance, for the parsing process in Figure 3.2(a), the splitting decision at the root span $(1, 5)$ results in a singleton span $(1, 1)$ and a general span $(2, 5)$. For this splitting decision, Eq. 3.2 requires the scores of $(1, 1)$ and $(2, 5)$. However, the set of pointing decisions $\mathcal{P}(T)$ does not cover the pointing for $(1, 1)$. This discrepancy can be resolved by modeling the singleton spans separately. To achieve that, we redefine Eq. 3.2 as follows:

$$s_{\text{split}}(i, k, j) = \begin{cases} sp(i \rightarrow i) + gp(i+1 \rightarrow j) & \text{if } k = i \\ gp(j-1 \rightarrow i) + sp(j \rightarrow j) & \text{if } k = j-1 \\ gp(k \rightarrow i) + gp(k+1 \rightarrow j) & \text{otherwise} \end{cases} \quad (3.4)$$

where sp and gp respectively represent the scores for the singleton and general pointing functions (to be defined formally in Section 3.3.3).

Remark on structural consistency. It is important to note that since the pointing functions are defined to have a global structural property (*i.e.*, the largest span that starts/ends with i), our model inherently enforces structural consistency. The pointing formulation of the parsing problem also makes the training process simple and efficient; it allows us to train the model effectively with simple cross entropy loss (see Section 3.3.4).

Label Assignment. Label assignment of spans is performed after every split decision. Specifically, as we split a span (i, j) into two sub-spans (i, k) and $(k+1, j)$ which corresponds to the pointing functions of $k \rightarrow i$ and $k+1 \rightarrow j$, we perform the label assignments for the two new sub-spans as

$$\begin{aligned} l_k &= \arg \max_{l \in L} gc(l|k) \\ l_{k+1} &= \arg \max_{l \in L} gc(l|k+1) \end{aligned} \quad (3.5)$$

where gc is the label classifier for any general (non-unary) span and L is the set of possible non-terminal labels. Following Shen et al. [63], we use a separate classifier uc for

FIGURE 3.2: Inferring the parse tree for a given sentence and its part-of-speech (POS) tags (predicted by an external POS tagger). Starting with the full sentence span $(1, 5)$ and its label S , we predict split point 1 using the base (sp) and general (gp) pointing scores as per Eqn. 3.2-3.4. The left singleton span $(1, 1)$ is assigned with a label NP and the right span $(2, 5)$ is assigned with a label $\emptyset$ using the label classifier gc as per Eqn. 3.5. The recursion of splitting and labeling continues until the process reaches a terminal node. The label assignment for the unary spans is done by the uc classifier.

determining the labels of the unary spans, e.g., the first layer of labels $NP, \emptyset, \dots, NP, \emptyset$) in Figure 3.2. Also, note that the label assignment is done based on only the query vector (the encoder state that is used to point).

Figure 3.2 illustrates the top-down parsing process for our running example. It consists of a sequence of pointing decisions (Figure 3.2(a), top to bottom), which are then trivially converted to the parse tree (Figure 3.2(b)). We also provide the pseudocode in Algorithm 2. Specifically, the algorithm finds the best split for the current span (i, j) using the pointing scores and pushes the newly created sub-spans into the FIFO queue $\mathcal{Q}$. The process terminates when there are no more spans to be split. Similar to Stern et al. [59], our parsing algorithm has the worst and best case time complexities of $\mathcal{O}(n^2)$ and $\mathcal{O}(n \log n)$, respectively.

Algorithm 2: Pointing Parsing Inference Algorithm

Require: Sentence length n ; pointing scores: $gp(i, j)$, $sp(i, j)$; label scores: $gc(l|i)$, $uc(l|i)$, $1 \leq i \leq j \leq n, l \in L_g/L_u$

Ensure: Parse tree $\mathcal{T}$

$\mathcal{Q} = [(1, n)]$ {queue of spans}

$\mathcal{S} = [(1, n, \arg \max_l gc(l|1))] \{ \text{general spans, labels} \}$

$\mathcal{U} = \{((t, t), \arg \max_l uc(l|t))\}_{t=1}^n \{ \text{unary spans, labels} \}$

while $\mathcal{Q} \neq \emptyset$ **do**

$(i, j) = \text{pop}(\mathcal{Q})$

if $j \leq i + 1$ **then**

Continue

end if

$k^* = \arg \max_{i \leq k < j} s_{\text{split}}(i, k, j) \{ \text{using } gp, sp \}$

if $k = i$ **then**

$\text{push}(\mathcal{Q}, (i + 1, j))$

$\text{push}(\mathcal{S}, (i + 1, j, \arg \max_l gc(l|i + 1)))$

else if $k = j - 1$ **then**

$\text{push}(\mathcal{Q}, (i, j - 1))$

$\text{push}(\mathcal{S}, (i, j - 1, \arg \max_l gc(l|j - 1)))$

else

$\text{push}(\mathcal{Q}, (i, k))$

$\text{push}(\mathcal{Q}, (k + 1, j))$

$\text{push}(\mathcal{S}, (i, k, \arg \max_l gc(l|k)))$

$\text{push}(\mathcal{S}, (k + 1, j, \arg \max_l gc(l|k + 1)))$

end if

end while

$\mathcal{T} = \mathcal{S} \cup \mathcal{U}$

3.3.3 Model Architecture

We now describe the architecture of our parsing model: the sentence encoder, the pointing model and the labeling model.

Sentence Encoder. Given an input sequence of n words $\mathbf{X} = (x_1, \dots, x_n)$, we first embed each word x_i to its respective vector representation e_i as:

$$e_i = e_i^{\text{char}} + e_i^{\text{word}} + e_i^{\text{pos}} \quad (3.6)$$

where e_i^{char} , e_i^{word} , e_i^{pos} are respectively the character, word, and part-of-speech (POS) embeddings of the word x_i . Following Kitaev and Klein [27], we use a character LSTM to compute the character embedding of a word. We experiment with both randomly initialized and pretrained word embeddings. If pretrained embeddings are used, the word

embedding e_i^{word} is the summation of the word's randomly-initialized embedding and the pretrained embedding. The POS embeddings (e_i^{pos}) are randomly initialized.

The word representations (e_i) are then passed to a neural network based sequence encoder to obtain their hidden representations. Since our method does not require any specific encoder, one may use any encoder model, such as Bi-LSTM [39] or self-attentive encoder [27]. In this chapter, unless otherwise specified, we use the self-attentive encoder model as our main sequence encoder because of its efficiency with parallel computation. The model is factorized into content and position information in both the self-attention sub-layer and the feed-forward layer. Details about this factorization process are provided in Kitaev and Klein [27].

Pointing and Labeling Models. The results of the aforementioned sequence encoding process are used to compute the pointing and labeling scores. More formally, the encoder network produces a sequence of n latent vectors $\mathbf{H} = (\mathbf{h}_1, \dots, \mathbf{h}_n)$ for the input sequence $\mathbf{X} = (x_1, \dots, x_n)$. After that, we apply four (4) separate position-wise two-layer Feed-Forward Networks (FFN), formulated as $\text{FFN}(x) = \text{ReLU}(xW_1 + b_1)W_2 + b_2$, to transform $\mathbf{H}$ into task-specific latent representations for the respective pointing and labeling tasks.

$$\mathbf{h}_i^{gp} = \text{FFN}_{gp}(\mathbf{h}_i); \quad \mathbf{h}_i^{sp} = \text{FFN}_{sp}(\mathbf{h}_i) \quad (3.7)$$

$$\mathbf{h}_i^{gc} = \text{FFN}_{gc}(\mathbf{h}_i); \quad \mathbf{h}_i^{uc} = \text{FFN}_{uc}(\mathbf{h}_i) \quad (3.8)$$

Note that there is no parameter sharing between FFN_{gp} , FFN_{sp} , FFN_{gc} and FFN_{uc} . The pointing functions are then modeled as the multinomial (or attention) distributions over the input indices for each input position i as follows.

$$gp(i, k) = \frac{\exp(\mathbf{h}_i^{gp}(\mathbf{h}_k^{gp})^T)}{\sum_{k=1}^n \exp(\mathbf{h}_i^{gp}(\mathbf{h}_k^{gp})^T)} \quad (3.9)$$

$$sp(i, k) = \frac{\exp(\mathbf{h}_i^{sp}(\mathbf{h}_k^{sp})^T)}{\sum_{k=1}^n \exp(\mathbf{h}_i^{sp}(\mathbf{h}_k^{sp})^T)} \quad (3.10)$$

For label assignment functions, we simply feed the label representations $\mathbf{H}^{gc} = (\mathbf{h}_1^{gc}, \dots, \mathbf{h}_n^{gc})$ and $\mathbf{H}^{uc} = (\mathbf{h}_1^{uc}, \dots, \mathbf{h}_n^{uc})$ into the respective softmax classification layers as follows.

$$gc(l|i) = \frac{\exp(\mathbf{h}_i^{gc} \mathbf{w}_l^{gc})}{\sum_{l=1}^{|L_g|} \exp(\mathbf{h}_i^{gc} \mathbf{w}_l^{gc})} \quad (3.11)$$

$$uc(l|i) = \frac{\exp(\mathbf{h}_i^{uc} \mathbf{w}_l^{uc})}{\sum_{l=1}^{|L_u|} \exp(\mathbf{h}_i^{uc} \mathbf{w}_l^{uc})} \quad (3.12)$$

where L_g and L_u are the set of possible labels for the general and unary spans respectively, $\mathbf{w}_l^{gc}$ and $\mathbf{w}_l^{uc}$ are the class-specific trainable weight vectors.

3.3.4 Training Objective

We train our parsing model by minimizing the total loss $\mathcal{L}_{total}(\theta)$ defined as:

$$\begin{aligned} \mathcal{L}_{total}(\theta) = & \mathcal{L}_{gp}(\theta_e, \theta_{gp}) + \mathcal{L}_{sp}(\theta_e, \theta_{sp}) \\ & + \mathcal{L}_{gc}(\theta_e, \theta_{gc}) + \mathcal{L}_{uc}(\theta_e, \theta_{uc}) \end{aligned} \quad (3.13)$$

where each individual loss is a cross entropy loss computed for the corresponding labeling or pointing task, and $\theta = \{\theta_e, \theta_{gp}, \theta_{sp}, \theta_{gc}, \theta_{uc}\}$ represents the overall model parameters; specifically, θ_e denotes the encoder parameters shared by all components, while $\theta_{gp}, \theta_{sp}, \theta_{gc}$ and θ_{uc} denote the separate parameters catering for the four pointing and labeling functions, gp, sp, gc and uc , respectively.

3.4 Experiments

To show the effectiveness of our approach, we conduct experiments on English and Multilingual parsing tasks. For English, we use the standard Wall Street Journal (WSJ) part of the Penn Treebank (PTB) [88], whereas for multilingual, we experiment with seven (7) different languages from the SPMRL 2013-2014 shared task [89]: Basque, French, German, Hungarian, Korean, Polish and Swedish.

For evaluation on PTB, we report the standard labeled precision (LP), labeled recall (LR), and labelled F1 (LF) computed by `evalb`². For the SPMRL datasets, we report labeled F1 and use the same setup in `evalb` as Kitaev and Klein [27].

3.4.1 English (PTB) Experiments

Setup. We follow the standard train/valid/test split, which uses sections 2-21 for training, section 22 for development and section 23 for evaluation. This gives 39,832 sentences for training, 1,700 sentences for development, and 2,416 sentences for testing. Following previous studies, our model uses POS tags predicted by the Stanford tagger [90].

For our model, we adopt the self-attention encoder with similar hyperparameter details proposed by Kitaev and Klein [27]. The character embeddings are of 64 dimensions. For general and unary label classifiers (*gc* and *uc*), the hidden dimension of the specific position-wise feed-forward networks is 250, while those for pointing functions (*gp* and *sp*) have hidden dimensions of 1024. Our model is trained using the Adam optimizer [91] with a batch size of 100 sentences. Additionally, we use 100 warm-up steps, within which we linearly increase the learning rate from 0 to the base learning rate of 0.008. Model selection for testing is performed based on the labeled F1 score on the validation set.

Results for Single Models. The experimental results on PTB for the models without pre-training are shown in Table 3.1. As it can be seen, our model achieves an F1 of 92.78, the highest among the models using *top-down inference* strategies. Specifically, our method outperforms Stern et al. [59] and Shen et al. [63] by about 1.0 point in F1-score. Notably, our model with LSTM encoder achieves an F1 of 92.26, which is still better than all the top-down parser methods.

On the other hand, while Kitaev and Klein [27] and Zhou and Zhao [68] achieve higher F1 score, their inference speed is significantly slower than ours because of the use of CKY based algorithms, which run at $\mathcal{O}(n^3)$ time complexity for Kitaev and Klein [27] and $\mathcal{O}(n^5)$ for Zhou and Zhao [68]. Furthermore, their training objectives involve the use of structural hinge loss, which requires online CKY inference during training. This makes their training

²<http://nlp.cs.nyu.edu/evalb/>

Model	LR	LP	F1
Top-Down Inference			
Stern et al. [59]	93.20	90.30	91.80
Shen et al. [63]	92.00	91.70	91.80
Our Model	92.81	92.75	92.78
CKY/Chart Inference			
Gaddy et al. [26]	-	-	92.10
Kitaev and Klein [27]	93.20	93.90	93.55
Other Approaches			
Gómez and Vilares [62]	-	-	90.7
Liu and Zhang [23]	-	-	91.8
Stern et al. [53]	92.57	92.56	92.56
Zhou and Zhao [68]	93.64	93.92	93.78

TABLE 3.1: Results for single models (no pre-training) on the PTB WSJ test set, Section 23.

time considerably slower than that of our method, which is trained directly with span-wise cross entropy loss.

In addition, Zhou and Zhao [68] uses external supervision (*head* information) from the dependency parsing task. Dependency parsing models, in fact, have a strong resemblance to the pointing mechanism that our model employs [48]. As such, integrating dependency parsing information into our model may also be beneficial. We leave this for future work.

Results with Pre-training Similar to Kitaev and Klein [27] and Kitaev et al. [28], we also evaluate our models with BERT [80] embeddings. Following them in the inclusion of contextualized token representations, we adjust the number of self-attentive layers to 2 and the base learning rate to 0.00005.

As shown in Table 3.2, our model achieves an F1 score of 95.48, which is on par with the state-of-the-art models. However, the advantage of our method is that it is faster than those methods. Specifically, our model runs at $\mathcal{O}(n^2)$ worst-case time complexity, while that of Kitaev et al. [28] is $\mathcal{O}(n^3)$. Comparison on parsing speed is discussed in the following section.

Parsing Speed Comparison. In addition to parsing performance in F1 scores, we also compare our parser against the previous neural approaches in terms of parsing speed. We record the parsing timing over 2416 sentences of the PTB test set with batch size of 1, on

Model	F1
Our model BERT _{BASE} -uncased	95.34
Our model BERT _{LARGE} -cased	95.48
Kitaev and Klein [27] ELMO	95.13
Kitaev et al. [28] BERT _{LARGE} -cased	95.59

TABLE 3.2: Results on PTB WSJ test set with pretraining.

a machine with NVIDIA GeForce GTX 1080Ti GPU and Intel(R) Xeon(R) Gold 6152 CPU. This setup is comparable to the setup of Shen et al. [63].

Model	# sents/sec
Petrov and Klein [57]	6.2
Zhu et al. [92]	89.5
Liu and Zhang [23]	79.2
Stern et al. [59]	75.5
Kitaev and Klein [27]	94.40
Shen et al. [63]	111.1
Our model	130.2

TABLE 3.3: Parsing speed for different models computed on the PTB WSJ test set.

As shown in Table 3.3, our parser outperforms Shen et al. [63] by 19 more sentences per second, despite the fact that our parsing algorithm runs at $\mathcal{O}(n^2)$ worse-case time complexity while the one used by Shen et al. [63] can theoretically run at $\mathcal{O}(n \log n)$ time complexity. To elaborate further, the algorithm presented in Shen et al. [63] can only run at $\mathcal{O}(n^2)$ complexity. To achieve $\mathcal{O}(n \log n)$ complexity, it needs to sort the list of syntactic distances, which the provided code³ does not implement. In addition, the speed up for our method can be attributed to the fact that our algorithm (see Algorithm 2) uses a *while loop*, while the algorithm of Shen et al. [63] has many recursive function calls. Recursive algorithms tend to be less empirically efficient than their equivalent while/for loops in handling low-level memory allocations and function call stacks.

³<https://github.com/hantek/distance-parser>

3.4.2 SPMRL Multilingual Experiments

Setup. Similar to the English PTB experiments, we use the predicted POS tags from external taggers (provided in the SPMRL datasets). The train/valid/test split is reported in Table 3.4. For single model evaluation, we use the identical hyper-parameters and optimizer setups as in English PTB. For experiments with pre-trained models, we use the multilingual BERT [80], which was trained jointly on 104 languages.

Language	Train	Valid	Test
Basque	7,577	948	946
French	14,759	1,235	2,541
German	40,472	5,000	5,000
Hebrew	5,000	500	716
Hungarian	8,146	1,051	1,009
Korean	23,010	2,066	2,287
Polish	6,578	821	822
Swedish	5,000	494	666

TABLE 3.4: SPMRL Multilingual dataset split.

Results. The results for the single models are reported in Table 3.5. We see that our model achieves the highest F1 score in Basque and Swedish, which are higher than the baselines by 0.52 and 1.37 respective in F1. Our method also performs competitively with the previous state-of-the-art methods on other languages.

Table 3.6 reports the performance of the models using pre-trained BERT. Evidently, our method achieves state-of-the-art results in Basque and Swedish, and performs on par with the previous best method by Kitaev et al. [28] in the other five languages. Again, note that our method is considerably faster and easier to train than the method of Kitaev et al. [28] due to the CKY inference.

Model	Basque	French	German	Hebrew	Hungarian	Korean	Polish	Swedish
Bjorkelund et al. [93]	88.24	82.53	81.66	89.80	91.72	83.81	90.50	85.50
Coavoux and Crabbé [94]	88.81	82.49	85.34	89.87	92.34	86.04	93.64	84.0
Kitaev and Klein [27]	89.71	84.06	87.69	90.35	92.69	86.59	93.69	84.45
Our Model	90.23	82.20	84.91	90.63	91.07	85.36	93.99	86.87

TABLE 3.5: SPMRL experiment single model test.

Model	Basque	French	German	Hebrew	Hungarian	Korean	Polish	Swedish
Kitaev et al. [28]	91.63	87.43	90.20	92.99	94.90	88.80	96.36	88.86
Our model	92.02	86.69	90.28	93.67	94.24	88.71	96.14	89.10

TABLE 3.6: SPMRL experiment pre-trained model test (with pretraining).

3.5 Summary

We have presented a novel constituency parsing method that is based on a pointing mechanism. Our method utilizes an efficient top-down decoding algorithm that uses pointing functions for scoring possible spans. The pointing formulation inherently captures global structural properties and allows efficient training with cross entropy loss. With experiments we have shown that our method outperforms all existing top-down methods on the English Penn Treebank parsing task. Our method with pre-training rivals the state-of-the-art method, while being faster than it. On multilingual constituency parsing, it also establishes new state-of-the-art in Basque and Swedish.

It is important to note that since the pointing functions are defined to have a global structural property, our method can overcome the need to consider all possible trees and be comparable to the global-chart parsing methods. However, our tree inference is still top-down greedy and “non-conditional” (the decoding history is not considered fully). In the next chapter, we will propose a method which can prevail over these two limitations: it considers previous splitting decisions and enhances the search space for high probable trees.

Chapter 4

A Conditional Splitting Framework for Efficient Constituency Parsing

[Chapter 3](#) formulates constituency parsing as a classical span prediction problem. We use neural pointing distribution from word to word to score each possible span. While effective, it still has the drawback of the top down parser: greedy and non-conditional.

In this chapter, we introduce a generic seq2seq parsing framework that casts constituency parsing problems into a series of conditional splitting decisions. Our parsing model estimates the conditional probability distribution of possible splitting points in a given text span and supports efficient top-down decoding, which is linear in number of nodes. The conditional splitting formulation together with efficient beam search inference facilitate structural consistency without relying on expensive structured inference. Experiments show that our model achieves good results on the standard constituency parsing tasks under settings with/without pre-trained representations and rivals state-of-the-art (SoTA) methods that are more computationally expensive than ours.¹

4.1 Introduction

Our parsing method in [Chapter 3](#) belongs to a category of greedy top-down parsers. This category of methods have received much attention lately due to their efficient inference.

¹This chapter is published as a portion of *A Conditional Splitting Framework for Efficient Constituency Parsing*, Proceedings of **ACL 2021** [95]

Their running time is usually $\mathcal{O}(n^2)$ [59, 63, 79]. However, they still suffer from exposure bias during the top-down inference, where one incorrect splitting step may affect subsequent steps because they are greedy and “non-conditional”. In this chapter, we propose a generic top-down neural framework for constituency parsing that overcomes these limitations. Our main contributions are:

- We cast the constituency parsing task into a series of conditional splitting decisions and use a seq2seq architecture to model the splitting decision at each decoding step. Our parsing model, which is an instance of a Pointer Network [4], estimates the pointing score from a span to a splitting boundary point, representing the likelihood that the span will be split at that point and create two child spans.
- The conditional probabilities of the splitting decisions are optimized using a cross entropy loss and structural consistency is maintained through a global pointing mechanism. The training process can be fully parallelized without requiring structured inference as in [62, 63, 79].
- Our model enables efficient top-down decoding with $\mathcal{O}(n)$ running time like transition-based parsers, while also supporting a customized beam search to get the best tree by searching through a reasonable search space of high scoring trees. The beam-search inference along with the structural consistency from the modeling makes our approach competitive with existing structured chart methods for constituency [27, 60]. Moreover, our parser does not rely on any handcrafted features (not even part-of-speech tags), which makes it more efficient and be flexible to different domains or languages.

In the experiments with English Penn Treebank, our model without pre-trained representations achieves 93.8 F1, outperforming all existing methods with similar time complexity. With pre-training, our model pushes the F1 score to 95.7, which is on par with the SoTA while supporting faster decoding with a speed of over 1,100 sentences per second (fastest so far). Our model also performs competitively with SoTA methods on the multilingual parsing tasks in the SPMRL 2013/2014 shared tasks.

4.2 Related Work

With the recent popularity of neural architectures, such as LSTMs [39] and Transformers [87], various neural models have been proposed to encode the input sentences and infer their constituency trees. To enforce structural consistency, such methods employ either a greedy transition-based, a globally optimized chart parsing, or a greedy top-down algorithm. Transition-based parsers [21–23] are popular for convenient computation. The ability to apply neural generation mechanism in generating the tree structure bottom-up attracts much attention from parsing community. However, these approaches still depend on the local decision which can result in the inconsistent tree prediction. Chart based methods [26, 27] or even top-down methods [59, 79] define score over phrases/span by using bidirectional neural network and utilizing token boundary-based representations of them. Then they use the computed span scores to mimic the PCFG rule that split a parent span to its two child spans. The chart-based parsing methods [26, 27] usually use CKY or dynamic programming to obtain the highest scoring tree in depth first bottom manners. The inference algorithm allows them to get high accurate tree while being slow. On the other hand, [59, 79] split tree in top-down order by finding highest splitting score in each step. Generally, they are faster than chart-based parsing methods but less accurate because of being greedy.

Meanwhile, researchers also tried to cast the parsing problem into tasks that can be solved differently. For example, Gómez and Vilares [62], Shen et al. [63] proposed to map the constituency tree of a sentence containing n tokens into a sequence of $n - 1$ labels or scalars. However, parsers of this type suffer from the exposure bias during inference. Beside these methods, Seq2Seq models have been used to generate a linearized form of the tree [64–67]. However, these methods may generate invalid trees when the open and end brackets do not match.

Our approach differs from previous methods in that it represents the constituency structure as a series of splitting representations. For that, we are free from scoring eligible span and only need to focus the splitting mechanism to build the tree structure. We use a Seq2Seq framework to model the conditional splitting decision at each step so that the current splitting decision can take into consideration of the splitting history. By enabling

FIGURE 4.1: An binarized constituency of the sentence “*She enjoys playing tennis.*”. Below the tree, we show the labeled span and splitting representations.

beam search, our model can find the best trees without the need to perform an expensive global search.

Our splitting mechanism shares some similarities with Pointer Network [4, 48, 96], but is distinct from it that in each decoding step, our method identifies the splitting point of a span and generates a new input for future steps instead of pointing to generate the next decoder input.

4.3 Parsing as a Splitting Problem

Constituency parsing can be considered as the problem of finding a set of labeled spans over the input text [59]. Let $\mathbb{S}(T)$ denote the set of *labeled spans* for a parse tree T , which can formally be expressed as (excluding the trivial singleton span layer):

$$\mathbb{S}(T) := \{((i_t, j_t), l_t)\}_{t=1}^{|\mathbb{S}(T)|} \text{ for } i_t < j_t \quad (4.1)$$

where l_t is the label of the text span (i_t, j_t) encompassing tokens from index i_t to index j_t .

Previous approaches to constituency parsing [27, 59, 79] train a neural model to score each possible span and then apply a greedy or dynamic programming algorithm to find the parse tree. In other words, these methods are span-based formulation.

In contrary, we formulate constituency parsing as the problem of finding the splitting points in a recursive, top-down manner. For each parent node in a tree that spans over (i, j) , our parsing model is trained to point to the *boundary* between the tokens at k and $k + 1$ positions to split the parent span into two child spans (i, k) and $(k + 1, j)$. This is done through the Pointing mechanism [4], where each splitting decision is modeled as a multinomial distribution over the input elements, which in our case are the token boundaries.

The correspondence between token- and boundary-based representations of a tree is straight-forward. After including the start-of-sentence ($\langle \text{sos} \rangle$) and end-of-sentence ($\langle \text{eos} \rangle$) tokens, the token-based span (i, j) is equivalent to the boundary-based span $(i - 1, j)$ and the boundary between i -th and $(i + 1)$ -th tokens is indexed as i . For example, the (boundary-based) span “enjoys playing tennis” in Figure 4.1 is defined as $(1, 4)$. Similarly, the boundary between the tokens ‘enjoys’ and ‘playing’ is indexed with 2.²

Following the common practice in constituency parsing, we binarize the n -ary tree by introducing a dummy label $\emptyset$. We also collapsed the nested labeled spans in the unary chains into unique atomic labels, such as S-VP in Figure 4.1. Every span represents an internal node in the tree, which has a left and a right child. Therefore, we can represent each internal node by its split into left and right children. Based on this, we define the set of splitting decisions $\mathbb{C}(T)$ for a constituency tree T as follows.

Proposition 2. A binary constituency tree T of a sentence containing n tokens can be transformed into a set of splitting decisions $\mathbb{C}(T) = \{(i, j) \rightarrow k : i < k < j\}$ such that the parent span (i, j) is split into two child spans (i, k) and (k, j) .

An example of the splitting representation of a tree is shown in Figure 4.1 (without the node labels). Note that our transformed representation has a one-to-one mapping with the tree since each splitting decision corresponds to one and only one internal node in the tree. We follow a depth-first order of the decision sequence, which in our preliminary experiments showed more consistent performance than other alternatives like breadth-first order.

²We use the same example from [59, 63, 79] to distinguish the differences between the methods.

by conditioning on the current input span (i_t, j_t) and previous splitting decisions $(i, j) \rightarrow k)_{<t}$. This conditional splitting formulation (decision at step t depends on previous steps) can help our model to find better trees compared to *non-conditional* top-down parsers [59, 63, 79], thus bridging the gap between the global (but expensive) and the local (but efficient) models. The labels $L(T)$ can be modeled by using a label classifier, as described later in the next section.

4.4.1 Model Architecture

We now describe the components of our parsing model: the sentence encoder, the span representation, the pointing model and the labeling model.

Sentence Encoder Given an input sequence of n tokens $\mathbf{x} = (x_1, \dots, x_n)$, we first add $\langle \text{sos} \rangle$ and $\langle \text{eos} \rangle$ markers to the sequence. After that, each token t in the sequence is mapped into its dense vector representation e_t as

$$e_t = [e_t^{\text{char}}, e_t^{\text{word}}] \quad (4.4)$$

where $e_t^{\text{char}}, e_t^{\text{word}}$ are respectively the character and word embeddings of token t . Similar to [27, 79], we use a character LSTM to compute the character embedding of a token. We experiment with both randomly initialized and pretrained token embeddings. When pretrained embedding is used, the character embedding is replaced by the pretrained token embedding. The token representations are then passed to a 3-layer Bi-LSTM encoder to obtain their contextual representations. In the experiments, we find that even without the POS-tags, our model performs competitively with other baselines that use them.

Boundary and Span Representations To represent each boundary between positions k and $k + 1$, we use the fencepost representation [22, 59]:

$$h_k = [f_k, b_{k+1}] \quad (4.5)$$

FIGURE 4.3: Illustration of our boundary-based span encoder. Here we have shown the representation for the boundary at 1 and the representation of the boundary-based span $(0, 5)$ that corresponds to the sentence “*She enjoys playing tennis .*”.

where $\mathbf{f}_k$ and $\mathbf{b}_{k+1}$ are the forward and backward LSTM hidden vectors at positions k and $k + 1$, respectively. To represent the span (i, j) , we compute a linear combination of the two endpoints

$$\mathbf{h}_{i,j} = \mathbf{W}_1 \mathbf{h}_i + \mathbf{W}_2 \mathbf{h}_j \quad (4.6)$$

This span representation will be used as input to the decoder. Figure 4.3 shows the boundary-based span representations for our example.

The Decoder Our model uses a unidirectional LSTM as the decoder. At each decoding step t , the decoder takes as input the corresponding span (i, j) (specifically, $\mathbf{h}_{i,j}$) and its previous state $\mathbf{d}_{t-1}$ to generate the current state $\mathbf{d}_t$ and then apply a biaffine function [97] between $\mathbf{d}_t$ and **all** of the encoded boundary representations $(\mathbf{h}_0, \mathbf{h}_1, \dots, \mathbf{h}_n)$ as follows:

$$\mathbf{d}'_t = \text{MLP}_d(\mathbf{d}_t) \quad \mathbf{h}'_i = \text{MLP}_h(\mathbf{h}_i) \quad (4.7)$$

$$s_{t,i} = \mathbf{d}'_t{}^T \mathbf{W}_{dh} \mathbf{h}'_i + \mathbf{h}'_i{}^T \mathbf{w}_h \quad (4.8)$$

$$a_{t,i} = \frac{\exp(s_{t,i})}{\sum_{i=1}^n \exp(s_{t,i})} \quad (4.9)$$

where each MLP operation includes a linear transformation with LeakyReLU activation to transform $\mathbf{d}$ and $\mathbf{h}$ into equal-sized vectors, and $\mathbf{W}_{dh} \in \mathbb{R}^{d \times d}$ and $\mathbf{w}_h \in \mathbb{R}^d$ are respectively the weight matrix and weight vector for the biaffine function. The biaffine scores are then passed through a softmax layer to acquire the pointing distribution $\mathbf{a}_t \in [0, 1]^n$

for the splitting decision. When decoding the tree during inference, at each step we only examine the ‘valid’ splitting points between i and j , it is $i < k < j$.

Label Classifier For constituency parsing, we perform the label assignments for a span (i, j) as:

$$\mathbf{h}_i^l = \text{MLP}_l(\mathbf{h}_i); \quad \mathbf{h}_j^r = \text{MLP}_r(\mathbf{h}_j) \quad (4.10)$$

$$P_\theta(l|i, j) = \text{softmax}((\mathbf{h}_i^l)^T \mathbf{W}_{lr} \mathbf{h}_j^r + (\mathbf{h}_i^l)^T \mathbf{W}_l + (\mathbf{h}_j^r)^T \mathbf{W}_r + \mathbf{b}) \quad (4.11)$$

$$l_{i,j} = \arg \max_{l \in L} P_\theta(l|i, j) \quad (4.12)$$

where each of MLP_l and MLP_r includes a linear transformation with LeakyReLU activations to transform the left and right spans into equal-sized vectors, and $\mathbf{W}_{lr} \in \mathbb{R}^{d \times L \times d}$, $\mathbf{W}_l \in \mathbb{R}^{d \times L}$, $\mathbf{W}_r \in \mathbb{R}^{d \times L}$ are the weights and $\mathbf{b}$ is a bias vector with L being the number of phrasal labels.

Training Objective The total loss is simply the sum of the cross entropy losses for predicting the structure (split decisions) and the labels:

$$\mathcal{L}_{\text{total}}(\theta) = \mathcal{L}_{\text{split}}(\theta_e, \theta_d) + \mathcal{L}_{\text{label}}(\theta_e, \theta_{\text{label}}) \quad (4.13)$$

where $\theta = \{\theta_e, \theta_d, \theta_{\text{label}}\}$ denotes the overall model parameters, which includes the encoder parameters θ_e shared by all components, parameters for splitting θ_d and parameters for labeling θ_{label} .

4.4.2 Top-Down Beam-Search Inference

As mentioned, existing top-down constituency parsers do not consider the decoding history. They also perform greedy inference. With our conditional splitting formulation, our method can not only model the splitting history but also enhance the search space of high scoring trees through beam search.

Algorithm 3: Constituency Tree Inference with Beam Search

Require: Sentence length n ; beam width B ; boundary-based encoder states: $(\mathbf{h}_0, \mathbf{h}_1, \dots, \mathbf{h}_n)$; label scores: $P_\theta(l|i, j)$, $0 \leq i < j \leq n$, $l \in \{1, \dots, L\}$, initial decoder state $\mathbf{s}$.

Ensure: Parse tree T

- 1: $L_d = n - 1$ {Decoding length}
- 2: beam = array of L_d items {List of empty beam items}
- 3: init_tree = $[(0, n), (0, 0), \dots, (0, 0)]$ $\{n - 2$ paddings $(0, 0)\}$
- 4: beam[0] = $(0, \mathbf{s}, \text{init_tree})$ {Init 1st item(log-prob, state, tree)}
- 5: **for** $t = 1$ **to** L_d **do**
- 6: **for** $(\log p, \mathbf{s}, \text{tree}) \in \text{beam}[t - 1]$ **do**
- 7: $(i, j) = \text{tree}[t - 1]$ {Current span to split}
- 8: $\mathbf{a}, \mathbf{s}' = \text{decoder-step}(\mathbf{s}, \mathbf{h}_{i,j})$ { $\mathbf{a}$: split prob. dist.}
- 9: **for** $(k, p_k) \in \text{top-}B(\mathbf{a})$ **and** $i < k < j$ **do**
- 10: curr-tree = tree
- 11: **if** $k > i + 1$ **then**
- 12: curr-tree[t] = (i, k)
- 13: **end if**
- 14: **if** $j > k + 1$ **then**
- 15: curr-tree[$t + j - k - 1$] = (k, j)
- 16: **end if**
- 17: push $(\log p + \log(p_k), \mathbf{s}', \text{curr-tree})$ to beam[t]
- 18: **end for**
- 19: **end for**
- 20: prune beam[t] {Keep top- B highest score trees}
- 21: **end for**
- 22: $\log p^*, \mathbf{s}^*, \mathcal{S}^* = \arg \max_{\log p} \text{beam}[L_d]$ $\{\mathcal{S}^*$: best structure}
- 23: labeled-spans = $[(i, j, \arg \max_l P_\theta(l|i, j))] \quad \forall (i, j) \in \mathcal{S}^*$
- 24: labeled-singletons = $[(i, i + 1, \arg \max_l P_\theta(l|i, i + 1))] \text{ for } i = \{0, \dots, n - 1\}$
- 25: $\mathcal{T} = \text{labeled-spans} \cup \text{labeled-singletons}$

At each step, our decoder points to *all* the encoded *boundary* representations which ensures that the pointing scores are in the same scale, allowing a fair comparison between the total scores of all candidate subtrees. With these uniform scores, we could apply a beam search to infer the most probable tree using our model. Specifically, the method generates the tree in depth-first order while maintaining top- B (beam size) partial trees at each step. It terminates exactly after $n - 1$ steps, which matches the number of internal nodes in the tree. Each decoding step can be parallelized ($\mathcal{O}(1)$ complexity) using GPUs. This makes our algorithm run at $\mathcal{O}(n)$ steps, which is faster than most top-down methods.

Our method can find the best tree by using beam search to compare high scoring trees within a reasonable search space, making our model competitive with existing structured (globally) inference methods that use more expensive algorithms like CKY and/or larger models [27, 60].

4.5 Experiments

Datasets and Metrics To show the effectiveness of our approach, we conduct experiments on constituency parsing tasks. We use the standard Wall Street Journal (WSJ) part of the Penn Treebank (PTB) [88] for English and the multilingual parsing tasks on seven different languages from the SPMRL 2013-2014 shared task [89]: Basque, French, German, Hungarian, Korean, Polish and Swedish.

For evaluation on constituency parsing, we report the standard labeled precision (LP), labeled recall (LR), and labelled F1 computed by `evalb`³.

4.5.1 English (PTB) Constituency Parsing

Setup We follow the standard train/valid/test split, which uses Sections 2-21 for training, Section 22 for development and Section 23 for evaluation. This results in 39,832 sentences for training, 1,700 for development, and 2,416 for testing. For our model, we use an LSTM encoder-decoder framework with a 3-layer bidirectional encoder and 3-layer unidirectional decoder. The word embedding size is 100 while the character embedding size is 50; the LSTM hidden size is 400. The hidden dimension in MLP modules and biaffine function for split point prediction is 500. The beam width B is set to 20. We use the Adam optimizer [91] with a batch size of 5000 tokens, and an initial learning rate of 0.002 which decays at the rate 0.75 exponentially at every 5k steps. Model selection for final evaluation is performed based on the labeled F1 score on the development set.

Results without Pre-training From the results shown in Table 4.1, we see that our model achieves an F1 of 93.77, the highest among models that use *top-down* methods. Specifically, our parser outperforms Stern et al. [59], Shen et al. [63] by about 2 points in F1-score and Nguyen et al. [79] by ~ 1 point. Notably, without beam search (beam width 1 or greedy decoding), our model achieves an F1 of 93.40, which is still better than other top-down methods. Our model also performs competitively with CKY-based methods like [27, 60, 68, 98], while these methods run slower than ours. Besides, Zhou and Zhao [68]

³<http://nlp.cs.nyu.edu/evalb/>

Model	LR	LP	F1
Top-Down Inference			
Stern et al. [59]	93.20	90.30	91.80
Shen et al. [63]	92.00	91.70	91.80
Nguyen et al. [79]	92.91	92.75	92.78
Our Model	93.90	93.63	93.77
CKY/Chart Inference			
Gaddy et al. [26]	91.76	92.41	92.08
Kitaev and Klein [27]	93.20	93.90	93.55
Wei et al. [98]	93.3	94.1	93.7
Zhang et al. [60]	93.84	93.58	93.71
Other Approaches			
Gómez and Vilares [62]	-	-	90.7
Liu and Zhang [23]	-	-	91.8
Stern et al. [53]	92.57	92.56	92.56
Zhou and Zhao [68]	93.64	93.92	93.78

TABLE 4.1: Results for single models (no pre-training) on the PTB WSJ test set, Section 23.

also use external supervision from the dependency parsing task which our method does not use.

Results with Pre-training Similar to [27, 28], we also evaluate our parser with BERT embeddings [80]. They fine-tuned *Bert-large-cased* on the task, while in our work keeping it frozen was already good enough (gives training efficiency). As shown in Table 4.2, our model achieves an F1 of 95.7, which is on par with SoTA models. However, our parser runs faster than other methods. Specifically, our model runs at $\mathcal{O}(n)$ time complexity, while CKY needs $\mathcal{O}(n^3)$. Comprehensive comparisons on parsing speed are presented later.

Model	F1
Nguyen et al. [79]	95.5
Our model	95.7
Kitaev et al. [28]	95.6
Zhang et al. [60]	95.7
Wei et al. [98]	95.8
Zhou and Zhao [68]	95.8

TABLE 4.2: Results on PTB WSJ test set with pretraining.

Error Analysis. Fig. 4.4 shows an example where our model mistakenly labels *NP* as $\emptyset$. It looks simple but crucial because the dummy label $\emptyset$ affects not only the label but also the tree structure. To avoid this mistake, we need to change the pointing/splitting mechanism to allow n-ary splitting or combine the structure splitting and label into one single decoding step. We leave this to future work.

FIGURE 4.4: An error example where our system incorrectly labels a *NP* as $\emptyset$.

4.5.2 SPMRL Multilingual Constituency Parsing

We use the identical hyper-parameters and optimizer setups as in English PTB. We follow the standard train/valid/test split provided in the SPMRL datasets; details are reported in the Appendix.

From the results in Table 4.3, we see that our model achieves the highest F1 in French, Hungarian and Korean and higher than the best baseline by 0.06, 0.15 and 0.13, respectively. Our method also rivals existing SoTA methods on other languages even though some of them use predicted POS tags.

Model	Basque	French	German	Hungarian	Korean	Polish	Swedish
Bjorkelund et al. [93] ⁺	88.24	82.53	81.66	91.72	83.81	90.50	85.50
Coavoux and Crabbé [94] ⁺	88.81	82.49	85.34	92.34	86.04	93.64	84.0
Kitaev and Klein [27]	89.71	84.06	87.69	92.69	86.59	93.69	84.45
Nguyen et al. [79] ⁺	90.23	82.20	84.91	91.07	85.36	93.99	86.87
Our Model	89.74	84.12	85.21	92.84	86.72	92.10	85.81

TABLE 4.3: Results on SPMRL test sets without pre-training. The sign ⁺ denotes that systems use predicted POS tags.

System	Speed (Sents/s)	Speedup
Petrov and Klein [57] (Berkeley)	6	1.0x
Zhu et al. [92](ZPar)	90	15.0x
Stern et al. [59]	76	12.7x
Shen et al. [63]	111	18.5x
Nguyen et al. [79]	130	21.7x
Zhou and Zhao [68]	159	26.5x
Wei et al. [98]	220	36.7x
Kitaev and Klein [27]	332	55.3x
Gómez and Vilares [62]	780	130x
Zhang et al. [60]	924	154x
Our model	1127	187.3x

TABLE 4.4: Speed comparison of our parser with existing constituency parsers.

4.5.3 Parsing Speed Comparison

We compare parsing speed of different models in Table 4.4. We ran our models on both CPU (Intel Xeon W-2133) and GPU (Nvidia GTX 1080 Ti).

The Berkeley Parser and ZPar are two representative non-neural parsers without access to GPUs. Stern et al. [59] employ max-margin training and perform top-down greedy decoding on CPUs. Meanwhile, Kitaev and Klein [27], Zhou and Zhao [68], Wei et al. [98] use a self-attention encoder and perform decoding using Cython for acceleration. Zhang et al. [60] perform CKY decoding on GPU. The parser proposed by Gómez and Vilares [62] is also efficient as it treats parsing as a sequence labeling task. However, its parsing accuracy is much lower compared to others (90.7 F1 in Table 4.1).

We see that our parser is much more efficient than existing ones. It utilizes neural modules to perform splitting, which is optimized and parallelized with efficient GPU implementation. It can parse 1,127 sentences/second, which is faster than existing parsers. In fact, there is still room to improve our speed by choosing better architectures, like the Transformer which has $\mathcal{O}(1)$ running time in encoding a sentence compared to $\mathcal{O}(n)$ of the bi-LSTM encoder. Moreover, allowing tree generation by splitting the spans/nodes at the same tree level in parallel at each step can boost the speed further. We leave these extensions to future work.

4.6 Summary

We have presented a novel, generic parsing method for constituency parsing based on a Seq2Seq framework. Our method supports an efficient top-down decoding algorithm that

uses a pointing function for scoring possible splitting points. The pointing mechanism captures global structural properties of a tree and allows efficient training with a cross entropy loss. Through experiments we have shown that our method outperforms all existing top-down methods on English Penn Treebank and some of SPMRL Multilingual sentence-level parsing tasks. With pre-trained representations, our method rivals state-of-the-art methods, while being faster. Our model so far works well for constituency parsing. It is not straight-forward to apply our method to discourse parsing without modification in the model. It is because constituency parsing methods are designed to split the sentence until it reaches individual words, it would need an extra stopping mechanism to stop when it reaches a potential EDU (span of words). In the next chapter, we would extend our model to not only achieve good performance with discourse parsing given gold EDUs but also solve segmentation and parsing in a single embedded framework.

Chapter 5

RST Parsing from Scratch

[Chapter 4](#) introduced a conditional splitting framework for constituency parsing. In this chapter, we extend it further and introduce a novel top-down end-to-end formulation for text level discourse parsing in the Rhetorical Structure Theory (RST) framework. In this formulation, we consider discourse parsing as a sequence of splitting decisions at token boundaries and use a seq2seq network to model the splitting decisions. Our framework facilitates discourse parsing from scratch without requiring discourse segmentation as a prerequisite; rather, it yields segmentation as part of the parsing process. Our unified parsing model adopts a beam search to decode the best tree structure by searching through a space of high scoring trees. With extensive experiments on the standard RST discourse treebank, we demonstrate that our parser outperforms existing methods by a good margin in both end-to-end parsing and parsing with gold segmentation. More importantly, it does so without using any handcrafted features, making it faster and easily adaptable to new languages and domains.¹

5.1 Introduction

In a language, the clauses, sentences and paragraphs are logically connected together to form a coherent discourse. The goal of discourse parsing is to uncover this underlying coherence structure, which has been shown to benefit numerous NLP applications including

¹This chapter is published with materials from *RST Parsing from Scratch*, Proceedings of **NAACL 2021** [99] and *A Conditional Splitting Framework for Efficient Constituency Parsing*, Proceedings of **ACL 2021** [95]

text classification [100], summarization [101], sentiment analysis [102], machine translation evaluation [103] and conversational machine reading [104].

Rhetorical Structure Theory or RST [105], one of the most influential theories of discourse, postulates a hierarchical discourse structure called discourse tree (DT). The leaves of a DT are clause-like units, known as elementary discourse units (EDUs). Adjacent EDUs and higher-order spans are connected hierarchically through coherence relations (e.g., *Contrast*, *Explanation*). Spans connected through a relation are categorized based on their relative importance — *nucleus* being the main part, with *satellite* being the subordinate one. Fig. 5.1 exemplifies a DT. Finding discourse structure generally requires breaking the text into EDUs (discourse segmentation) and linking the EDUs into a DT (discourse parsing).

Discourse parsers can be singled out by whether they apply a bottom-up or top-down procedure. Bottom-up parsers include transition-based models [29, 76, 106, 107] or globally optimized chart parsing models [7, 69]. The former constructs a DT by a sequence of shift and reduce decisions, and can parse a text in asymptotic running time that is linear in number of EDUs. However, the transition-based parsers make greedy local decisions at each decoding step, which could propagate errors into future steps. In contrast, chart parsers learn scoring functions for sub-trees and adopt a CKY-like algorithm to search for the highest scoring tree. These methods normally have higher accuracy but suffer from a slow parsing speed with a complexity of $\mathcal{O}(n^3)$ for n EDUs. The top-down parsers are relatively new in discourse [31, 32, 49]. These methods focus on finding splitting points in each iteration to build a DT. However, the local decisions could still affect the performance as most of the methods are still greedy.

Like most other fields in NLP, language parsing has also undergone a major paradigm shift from traditional feature-based statistical parsing to end-to-end neural parsing. Being able to parse a document end-to-end from scratch is appealing for several key reasons. First, it makes the overall development procedure easily adaptable to new languages, domains and tasks by surpassing the expensive feature engineering step that often requires more time and domain/language expertise. Second, the lack of an explicit feature extraction phase makes the training and testing (decoding) faster.

Because of the task complexity, it is only recently that neural approaches have started to outperform traditional feature-rich methods. However, successful document level neural parsers still rely heavily on handcrafted features [29–32]. Therefore, even though these methods adopt a neural framework, they are not “end-to-end” and do not enjoy the above mentioned benefits of an end-to-end neural parser.

Moreover, in existing methods (both traditional and neural), discourse segmentation is detached from parsing and treated as a prerequisite step. Therefore, the errors in segmentation affect the overall parsing performance [49]. In view of the limitations of existing approaches, in this work we propose an end-to-end top-down document level parsing model that:

- Can generate a discourse tree from scratch without requiring discourse segmentation as a prerequisite step; rather, it generates the EDUs as a by-product of parsing. Crucially, this novel formulation facilitates solving the two tasks in a single neural model. Our formulation is generic and works in the same way when it is provided with the EDU segmentation.
- Treats discourse parsing as a sequence of splitting decisions at token boundaries and uses a seq2seq pointer network [4] to model the splitting decisions at each decoding step. Importantly, our seq2seq parsing model can adopt beam search to widen the search space for the highest scoring tree, which to our knowledge is also novel for the parsing problem.
- Does not rely on any handcrafted features, which makes it faster to train or test, and easily adaptable to other domains and languages.
- Achieves the state of the art (SoTA) with an F_1 score of 46.6 in the Full (label+structure) metric for end-to-end parsing on the RST Discourse Treebank, which outperforms many parsers that use gold EDU segmentation. With gold segmentation, our model achieves a SoTA F_1 score of 50.2 (Full), outperforming the best existing system by 2.1 absolute points. More importantly, it does so without using any handcrafted features (not even part-of-speech tags).

5.2 Related Work

Discourse analysis has been a long-established problem in NLP. Prior to the neural tsunami in NLP, discourse parsing methods commonly employed statistical models with hand-crafted features [7, 69, 76, 108]. Even within the neural paradigm, most previous studies still rely on external features to achieve their best performances [29, 30, 106, 107, 109]. These parsers adopt a bottom-up approach, either transition-based or chart-based parsing.

Recently, top-down parsing has attracted more attention due to its ability to maintain an overall view of the input text. Inspired by the Stack-Pointer network [48] for dependency parsing, Lin et al. [49] first propose a seq2seq model for sentence-level parsing. Zhang et al. [31] extend this to the document level. Kobayashi et al. [32] adopt a greedy splitting mechanism for discourse parsing inspired by Stern et al. [59]’s work in constituency parsing. By using pretrained models/embeddings and extra features (*e.g.*, syntactic, text organizational features), these models achieve competitive results. However, their decoder infers a tree greedily.

Moreover, existing parsers receive the EDUs from a segmenter to build the discourse tree, which makes them susceptible to errors when the segmenter produces incorrect EDUs [7, 31, 49, 110, 111]. There are also attempts which model constituency and discourse parsing jointly [112] and do not need to perform EDU preprocessing. It is based on the finding that each EDU generally corresponds to a constituent in constituency tree, *i.e.*, discourse structure usually aligns with constituency structure. However, it has the drawback that it needs to build joint syntacto-discourse data set for training which is not easily adaptable to new languages and domains.

5.3 Model

Assuming that a document has already been segmented into EDUs, following the traditional approach, the corresponding discourse tree (DT) can be represented as a set of labeled constituents.

$$\mathbb{C} := \{((i_t, k_t, j_t), r_t) | i_t \leq k_t < j_t\}_{t=1}^m \quad (5.1)$$

Boundary-based splitting representation when EDUs are provided

$$\mathbb{S}_{\text{edu}} = \{(0, 44) \rightarrow 4, (4, 44) \rightarrow 25, (4, 25) \rightarrow 17, (25, 44) \rightarrow 37, (25, 37) \rightarrow 33\}$$

Boundary-based splitting representation for end-to-end parsing

$$\mathbb{S} = \{(0, 44) \rightarrow 4, \mathbf{(0, 4) \rightarrow 4}, (4, 44) \rightarrow 25, (4, 25) \rightarrow 17, \mathbf{(4, 17) \rightarrow 17}, \mathbf{(17, 25) \rightarrow 25}, (25, 44) \rightarrow 37, (25, 37) \rightarrow 33, \mathbf{(25, 33) \rightarrow 33}, \mathbf{(33, 37) \rightarrow 37}, \mathbf{(37, 44) \rightarrow 44}\}$$

FIGURE 5.1: A discourse tree for two sentences in the RST discourse treebank. The internal nodes (e.g., *Attribution*, *Contrast*) denote the coherence relations and the edge labels reflect the nuclearity of the child span. Below the tree, we show the sequence of splitting decisions $\mathbb{S}_{\text{edu}}$ when EDUs are provided and $\mathbb{S}$ when EDUs are not provided (end-to-end parsing). The **bold** splitting decision represents the final split of the span, forming an EDU.

where $m = |\mathbb{C}|$ is the number of internal nodes in the tree and r_t is the relation label between the discourse unit containing EDUs i_t through k_t and the one containing EDUs $k_t + 1$ through j_t . A common pre-processing procedure in discourse parsing is to binarize the trees [7, 49]. Similar to constituency tree, we map the multi-nuclear relations, which have more than two discourse units, into a right-branching binary tree.

Traditionally, in RST parsing, discourse segmentation is performed first to obtain the sequence of EDUs, which is followed by the parsing process to assemble the EDUs into a labeled tree. In other words, traditionally discourse segmentation and parsing have been considered as two distinct tasks that are solved by two different models.

On the contrary, in this work we take a radically different approach that directly starts with parsing the (unsegmented) document in a top-down manner and treats discourse segmentation as a special case of parsing that we get as a by-product. Importantly, this novel formulation of the problem allows us to solve the two problems in a single neural model. Our parsing model is generic and also works in the same way when it is fed with an EDU-segmented text. Before presenting the model architecture, we first formulate the problem as a splitting decision problem at the token level.

	0	1	2	3	4	41	42	43	44	
<sod>	But	he	added	:	...	months	.	"	<eod>	
0	1	2	3	4		42	43	44	45	

FIGURE 5.2: Relation between token-boundary (above) and token (below) representations. A token-boundary position k is located between the tokens at k and $k + 1$.

5.3.1 Parsing as a Splitting Decision Problem

We reformulate the discourse parsing problem from Eq. (5.1) as a sequence of splitting decisions at *token boundaries* (instead of EDUs). Specifically, the input text is first prepended and appended with the special start (<sod>) and end (<eod>) tokens, respectively. We define the token-boundary as the indexed position between two consecutive tokens. For example, the constituent spanning “But he added :” in Fig. 5.2 is defined as $(0, 4)$.

Following the standard practice, we convert the discourse tree by transforming each multi-nuclear constituent into a hierarchical right-branching binary sub-tree. Every internal node in the resulting binary tree will have a left and a right constituent, allowing us to represent it by its split into the left and right children. Based on this, we define the parsing as a set of splitting decisions $\mathbb{S}$ at token-boundaries by the following proposition:

Proposition 3. Given a binarized discourse tree for a document containing n tokens, the tree can be converted into a set of token-boundary splitting decisions $\mathbb{S} = \{(i, j) \rightarrow k \mid i < k \leq j\}$ such that the parent constituent (i, j) either gets split into two child constituents (i, k) and (k, j) for $k < j$, or forms a terminal EDU unit for $k = j$, i.e., the span will not be split further (i.e., marks segmentation).

Notice that $\mathbb{S}$ is a generalized formulation of RST parsing, which also includes the decoding of EDUs as a special case ($k = j$). It is quite straight-forward to change this formulation to the parsing scenario, where discourse segmentation (sequence of EDUs) is provided. Formally, in that case, the tree can be converted into a set of splitting decisions $\mathbb{S}_{\text{edu}} = \{(i, j) \rightarrow k \mid i < k < j\}$ such that the constituent (i, j) gets split into two constituents (i, k) and (k, j) for $k < j$, i.e., we simply omit the special case of $k = j$ as the EDUs are given. In other words, in our generalized formulation, discourse segmentation is just one extra step of parsing, and can be done top-down end-to-end.

FIGURE 5.3: Our discourse parser along with a few decoding steps for a given document. The input to the decoder at each step is the representation of the span to be split. We predict the splitting point using the biaffine function between the corresponding decoder state and the token-boundary encoder representations. The figure is for end-to-end parsing, where each EDU-corresponding span points to its right edge to mark the EDU. The coherence relations between the left and right spans are assigned using a label classifier after the (approximately) optimal tree structure is formed using beam search.

An example of our formalism of the parsing problem is shown in Fig. 5.1 for a discourse tree spanning over two sentences (44 tokens); for simplicity, we do not show the relation labels corresponding to the splitting decisions (marked by $\rightarrow$). Since each splitting decision corresponds to one and only one internal node in the tree, it guarantees that the transformation from the tree to $\mathbb{S}$ (and $\mathbb{S}_{\text{edu}}$) has a one-to-one mapping. Therefore, predicting the sequence of such splitting decisions is equivalent to predicting the discourse tree (DT).

Seq2Seq Parsing Model. In this work, we adopt a structure-then-label framework. Specifically, we factorize the probability of a DT into the probability of the tree structure and the probability of the relations (*i.e.*, the node labels) as follows:

$$P_{\theta}(\text{DT}|\mathbf{x}) = P_{\theta}(\mathbb{S}, \mathbb{L}|\mathbf{x}) = P_{\theta}(\mathbb{L}|\mathbb{S}, \mathbf{x})P_{\theta}(\mathbb{S}|\mathbf{x}) \quad (5.2)$$

where $\mathbf{x}$ is the input document, and $\mathbb{S}$ and $\mathbb{L}$ respectively denote the structure and labels of the DT. This formulation allows us to first infer the best tree structure (*e.g.*, using beam search), and then find the corresponding labels.

As discussed, we consider the structure prediction problem as a sequence of splitting decisions to generate the tree in a top-down manner. We use a seq2seq pointer network [4] to model the sequence of splitting decisions (Fig. 5.3). We adopt a depth-first order

of the decision sequence, which showed more consistent performance in our preliminary experiments than other alternatives, such as breath-first order.

First, we encode the tokens in a document $\mathbf{x} = (x_0, \dots, x_n)$ with a document encoder and get the token-boundary representations $(\mathbf{h}_0, \dots, \mathbf{h}_n)$. Then, at each decoding step t , the model takes as input an internal node (i_t, j_t) , and produces an output y_t (by pointing to the token boundaries) that represents the splitting decision $(i_t, j_t) \rightarrow k_t$ to split it into two child constituents (i_t, k_t) and (k_t, j_t) . For example, the initial span $(0, 44)$ in Fig. 5.1 is split at boundary position 4, yielding two child spans $(0, 4)$ and $(4, 44)$. If the span $(0, 4)$ is given as an EDU (*i.e.*, segmentation given), the splitting stops at $(0, 4)$, thus omitted in $\mathbb{S}_{\text{edu}}$ (Fig. 5.1). Otherwise, an extra decision $(0, 4) \rightarrow 4 \in \mathbb{S}$ needs to be made to mark the EDUs for end-to-end parsing. With this, the probability of $\mathbb{S}$ can be expressed as:

$$\begin{aligned} P_\theta(\mathbb{S}|\mathbf{x}) &= \prod_{y_t \in \mathbb{S}} P_\theta(y_t|y_{<t}, \mathbf{x}) \\ &= \prod_{t=1}^{|\mathbb{S}|} P_\theta\left((i_t, j_t) \rightarrow k_t | ((i, j) \rightarrow k)_{<t}, \mathbf{x}\right) \end{aligned}$$

This end-to-end conditional splitting formulation is the main novelty of our method and is in contrast to previous approaches which rely on offline-inferred EDUs from a separate discourse segmenter. Our formalism streamlines the overall parsing process, unifies the neural components seamlessly and smoothens the training process.

5.3.2 Model Architecture

In the following, we describe the components of our parsing model: the document encoder, the boundary and span representations, the decoding process through the decoder and the label classifier.

Document Encoder. Given an input document of n words $\mathbf{x} = (x_1, \dots, x_n)$, we first add $\langle \text{sod} \rangle$ and $\langle \text{eod} \rangle$ markers to the sequence. After that, each token x_i in the sequence is mapped into its dense vector representation $\mathbf{e}_i$ as: $\mathbf{e}_i = [\mathbf{e}_i^{\text{char}}, \mathbf{e}_i^{\text{word}}]$, where $\mathbf{e}_i^{\text{char}}$, and $\mathbf{e}_i^{\text{word}}$ are respectively the character and word embeddings of token x_i . For word

FIGURE 5.4: Illustration of token-boundary span encoder. The figure lays out an example representation for the boundary at 2 and the representation of the token-boundary span $(0, 44)$, which corresponds to the whole document.

embedding, we experiment with (i) randomly initialized, (ii) pretrained static embeddings (GloVe from Pennington et al. [113]). To represent the character embedding of a token, we apply a character bidirectional LSTM *i.e.*, Bi-LSTM [39] or pretrained contextualized embeddings (XLNet from Yang et al. [45]). These document encoder methods can apply on any arbitrary length documents. The token representations are then passed to a sequence encoder of a three-layer Bi-LSTM to obtain their forward f_i and backward b_i contextual representations.

Token-boundary Span Representations. To represent each token-boundary position k between token positions k and $k + 1$, we use the fencepost representation [22]:

$$h_k = [f_k; b_{k+1}] \quad (5.3)$$

where f_k and b_{k+1} are the forward and backward LSTM hidden vectors of positions k and $k + 1$ respectively, and $[\cdot; \cdot]$ is the concatenation operation.

Then, to represent the token-boundary span (i, j) , we use the linear combination of the two endpoints i and j as:

$$h_{i,j} = W_1 h_i + W_2 h_j \quad (5.4)$$

where W_1 and W_2 are trainable weights. These span representations will be used as input to the decoder or the label classifier. Fig. 5.4 illustrates an example boundary span representation.

The Decoder. Our model uses a unidirectional LSTM as the decoder. At each decoding step t , the decoder takes as input the corresponding span (i, j) (*i.e.*, $h_{i,j}$) and its previous LSTM state d_{t-1} to generate the current state d_t and then the biaffine function [97] is

applied between $\mathbf{d}_t$ and *all* the encoded token-boundary representations $(\mathbf{h}_0, \mathbf{h}_1, \dots, \mathbf{h}_n)$ as follows:

$$\mathbf{d}'_t = \text{MLP}_d(\mathbf{d}_t) \quad \mathbf{h}'_i = \text{MLP}_h(\mathbf{h}_i) \quad (5.5)$$

$$s_t^i = \mathbf{d}'_t{}^T \mathbf{W}_{dh} \mathbf{h}'_i + \mathbf{h}'_i{}^T \mathbf{w}_h \quad (5.6)$$

$$a_t^i = \frac{\exp(s_t^i)}{\sum_{i=0}^n \exp(s_t^i)} \quad \text{for } i = 0, \dots, n \quad (5.7)$$

where each MLP operation comprises a linear transformation with LeakyReLU activation [114] to transform $\mathbf{d}_i$ and $\mathbf{h}_i$ into equal-sized vectors $\mathbf{d}'_t, \mathbf{h}'_i \in \mathbb{R}^d$, and $\mathbf{W}_{dh} \in \mathbb{R}^{d \times d}$ and $\mathbf{w}_h \in \mathbb{R}^d$ are respectively the weight matrix and weight vector for the biaffine function. The resulting biaffine scores s_t^i are then fed into a *softmax* layer to acquire the pointing distribution $\mathbf{a}_t^i \in [0, 1]^{n+1}$ for the splitting decision. During inference, when decoding the tree at step t , we only examine the “valid” splitting points between i and j , and we look for k such that $i < k \leq j$.

Label Classifier. We perform label assignment after decoding the entire tree structure. Each assignment takes into account the splitting decision that generated it since the label represents the relation between the child spans. Specifically, for a constituent (i, j) that was split into two child constituents (i, k) and (k, j) , we determine the coherence relation between them as follows:

$$\mathbf{h}_{ik}^l = \text{MLP}_l([\mathbf{h}_i; \mathbf{h}_k]); \quad \mathbf{h}_{kj}^r = \text{MLP}_r([\mathbf{h}_k; \mathbf{h}_j]) \quad (5.8)$$

$$P_\theta(l|(i, k), (k, j)) = \text{softmax}\left((\mathbf{h}_{ik}^l)^T \mathbf{W}_{lr} \mathbf{h}_{kj}^r + (\mathbf{h}_{ik}^l)^T \mathbf{W}_l + (\mathbf{h}_{kj}^r)^T \mathbf{W}_r + \mathbf{b}\right) \quad (5.9)$$

$$l_{(i,k),(k,j)}^* = \arg \max_{l \in L} P_\theta(l|(i, k), (k, j)) \quad (5.10)$$

where L is the total number of labels (*i.e.*, coherence relations with nuclearity attached); each of MLP_l and MLP_r includes a linear transformation with LeakyReLU activation to transform the left and right spans into equal-sized vectors $\mathbf{h}_{ik}^l, \mathbf{h}_{kj}^r \in \mathbb{R}^d$; $\mathbf{W}_{lr} \in \mathbb{R}^{d \times L \times d}$, $\mathbf{W}_l \in \mathbb{R}^{d \times L}$, $\mathbf{W}_r \in \mathbb{R}^{d \times L}$ are the weights and $\mathbf{b}$ is a bias vector.

Training Objective. Our parsing model is trained by minimizing the total loss defined as:

$$\mathcal{L}(\theta_e, \theta_d, \theta_l) = \mathcal{L}_s(\theta_e, \theta_d) + \mathcal{L}_l(\theta_e, \theta_l) \quad (5.11)$$

where structure $\mathcal{L}_s$ and label $\mathcal{L}_l$ losses are cross-entropy losses computed for the splitting and labeling tasks respectively, and θ_e, θ_d and θ_l denote the encoder, decoder and labeling parameters.

5.3.3 Complete Discourse Parsing Models

Having presented the generic framework, we now describe how it can be easily adapted to the two parsing scenarios: (i) end-to-end parsing and (ii) parsing with EDUs. We also describe the incorporation of beam search for inference.

End-to-End Parsing. As mentioned, previous work for end-to-end parsing assumes a separate segmenter that provides EDU-segmented texts to the parser. Our method, however, is an end-to-end framework that produces both the EDUs as well as the parse tree in the same inference process. To guide the search better, we incorporate an inductive bias into our inference based on the finding that most sentences have a well-formed subtree in the document-level tree [69], *i.e.*, discourse structure tends to align with the text structure (sentence boundary in this case); for example, Joty et al. [7], Fisher and Roark [115] found that more than 95% of the sentences have a well-formed subtree in the RST discourse treebank.

Our goal is to ensure that each sentence corresponds to an internal node in the tree. This can be achieved by a simple adjustment in our inference. When decoding at time step t with the span (i_t, j_t) as input, if the span contains $M > 0$ sentence boundaries within it, we pick the one that has the highest pointing score (Eq. 5.7) among the M alternatives as the split point k_t . If there is no sentence boundary within the input span ($M = 0$), we find the next split point as usual. In other words, sentence boundaries in a document get the chance to be split before the token boundaries inside a sentence. This constraint is indeed similar to the 1S-1S (1 subtree for 1 sentence) constraint of Joty et al. [7]’s bottom-up parsing, and is also consistent with the property that EDUs are always within the sentence boundary. Algorithm 4 illustrate the end-to-end inference algorithm

Algorithm 4: Discourse Tree Inference (end-to-end)

Require: Document length n ; boundary encoder states: $(\mathbf{h}_0, \mathbf{h}_1, \dots, \mathbf{h}_n)$; sentence boundary set $\mathbb{SB}$; label scores: $P(l|(i, k), (k, j))$, $0 \leq i < k \leq j \leq n, l \in L$, initial decoder state st .

Ensure: Parse tree DT

$\mathcal{ST} = [(0, n)]$ {stack of spans}
 $\mathcal{S} = []$
while $\mathcal{ST} \neq \emptyset$ **do**
 $(i, j) = \text{pop}(\mathcal{ST})$ {Current span to split}
 $\mathbf{a}, st = \text{dec}(st, \mathbf{h}_{i,j})$ { $\mathbf{a}$: split prob. dist.}
 if $(i, j) \cap \mathbb{SB} \neq \emptyset$ **then**
 $k = \arg \max_{i < k \leq j \ \& \ k \in \mathbb{SB}} \mathbf{a}$
 else
 $k = \arg \max_{i < k \leq j} \mathbf{a}$
 end if
 if $j - 1 > k > i + 1$ **then**
 push($\mathcal{ST}, (k, j)$)
 push($\mathcal{ST}, (i, k)$)
 else if $j - 1 > k = i + 1$ **then**
 push($\mathcal{ST}, (k, j)$)
 else if $k = j - 1 > i + 1$ **then**
 push($\mathcal{ST}, (i, k)$)
 end if
 if $k \neq j$ **then**
 push($\mathcal{S}, ((i, k, j))$)
 end if
end while
 $DT = [((i, k, j), \arg \max_l P(l|(i, k)(k, j)) \forall (i, k, j) \in \mathcal{S})]$

Parsing with EDUs. When segmentation information is provided, we can have a better encoding of the EDUs to construct the tree. Specifically, rather than simply taking the token-boundary representation corresponding to the EDU boundary as the EDU representation, we adopt a hierarchical approach, where we add another Bi-LSTM layer (called “Boundary LSTM”) that connects EDU boundaries (a figure of this framework is in the Appendix). In other words, the input sequence to this LSTM layer is $(\bar{\mathbf{h}}_0, \dots, \bar{\mathbf{h}}_m)$, where $\bar{\mathbf{h}}_0 = \mathbf{h}_0$, $\bar{\mathbf{h}}_m = \mathbf{h}_n$ and $\bar{\mathbf{h}}_j \in \{\mathbf{h}_1, \dots, \mathbf{h}_{n-1}\}$ such that $\bar{\mathbf{h}}_j$ is an EDU boundary. For instance, for the example in Fig. 5.1, the input to the Boundary LSTM layer is $(\mathbf{h}_0, \mathbf{h}_4, \mathbf{h}_{17}, \mathbf{h}_{25}, \mathbf{h}_{33}, \mathbf{h}_{37}, \mathbf{h}_{44})$.

This hierarchical representation facilitates better modeling of relations between EDUs and higher order spans, and can capture long-range dependencies better, especially for long documents.

Algorithm 5: Discourse Tree Inference with Beam Search (with gold EDUs)

Require: Number of EDUs in document m ; beam width B ; EDU boundary-based encoder states: $(\bar{h}_0, \dots, \bar{h}_m)$; label scores: $P_\theta(l|(i, k), (k, j), 0 \leq i < k < j \leq m, l \in \{1, \dots, L\})$, initial decoder state $\mathbf{s}$.

Ensure: Parse tree DT

```

1:  $L_d = m - 1$  {Decoding length}
2: beam = array of  $L_d$  items {List of empty beam items}
3: init_input_span =  $[(0, m), (0, 0), \dots, (0, 0)]$  { $m - 2$  paddings (0,0)}
4: init_tree =  $[(0, 0, 0), (0, 0, 0), \dots, (0, 0, 0)]$  { $m - 1$  elementss (0,0,0)}
5: beam[0] =  $(0, \mathbf{s}, \text{init\_input\_span}, \text{init\_tree})$  {Init 1st item(log-prob,state,input\_span,tree)}
6: for  $t = 1$  to  $L_d$  do
7:   for  $(\text{logp}, \mathbf{s}, \text{input\_span}, \text{tree}) \in \text{beam}[t - 1]$  do
8:      $(i, j) = \text{input\_span}[t - 1]$  {Current span to split}
9:      $\mathbf{a}, \mathbf{s}' = \text{decoder\_step}(\mathbf{s}, \bar{\mathbf{h}}_{i,j})$  { $\mathbf{a}$ : split prob. dist.}
10:    for  $(k, p_k) \in \text{top-}B(\mathbf{a})$  and  $i < k < j$  do
11:      curr_input_span = input_span
12:      curr_tree = tree
13:      curr_tree[t - 1] =  $(i, k, j)$ 
14:      if  $k > i + 1$  then
15:        curr_input_span[t] =  $(i, k)$ 
16:      end if
17:      if  $j > k + 1$  then
18:        curr_input_span[t + j - k - 1] =  $(k, j)$ 
19:      end if
20:      push  $(\text{logp} + \log(p_k), \mathbf{s}', \text{curr\_input\_span}, \text{curr\_tree})$  to beam[t]
21:    end for
22:  end for
23:  prune beam[t] {Keep top- $B$  highest score trees}
24: end for
25:  $\text{logp}^*, \mathbf{s}^*, \mathbf{ip}^*, \mathbb{S}^* = \arg \max_{\text{logp}} \text{beam}[L_d]$  { $\mathbb{S}^*$ : best structure}
26:  $DT = [(i, k, j, \arg \max_l P_\theta(l|(i, k), (k, j)) \ \forall (i, k, j) \in \mathbb{S}^*)$ 

```

Incorporating Beam Search. Previous works [31, 49] which also use a seq2seq architecture, compute the pointing scores over the token or span representations only within the input span. For example, for an input span (i, j) , the pointing scores are computed considering only $(\mathbf{h}_i, \dots, \mathbf{h}_j)$ as opposed to $(\mathbf{h}_1, \dots, \mathbf{h}_n)$ in our Eq. 5.7. This makes the scales of the scores uneven across different input spans as the lengths of the spans vary. Thus, such scores cannot be objectively compared across sub-trees globally at the full-tree level. In addition, since efficient global search methods like beam search cannot be applied properly with non-uniform scores, these previous methods had to remain greedy at each decoding step. In contrast, our decoder points to all the encoded token-boundary representations in every step (Eq. 5.7). This ensures that the pointing scores are evenly scaled, allowing fair comparisons between the scores of all candidate sub-trees. Therefore, our method enables the effective use of beam search through highly probable candidate trees.

We conduct our experiments on discourse parsing with and without gold segmentation. We use the standard RST Discourse Treebank or RST-DT [116] for training and evaluation at both document and sentence levels. At document level, it consists of 385 annotated Wall Street Journal news articles: 347 for training and 38 for testing. At sentence

level, it has 7321 discourse trees for training and 951 for testing. We randomly select 10% of the training set as our development set for hyper-parameter tuning. For evaluation, we report the standard metrics Span, Nuclearity, Relation and Full F1 scores, computed using the standard Parseval [10, 117] and RST-Parseval [78] metrics.

5.4 Experiments

We conduct our experiments on discourse parsing with and without gold segmentation. We use the standard English RST Discourse Treebank or RST-DT [116] for training and evaluation. It consists of 385 annotated Wall Street Journal news articles: 347 for training and 38 for testing. We randomly select 10% of the training set as our development set for hyper-parameter tuning. Following prior work, we adopted the same 18 courser relations defined in [118]. For evaluation, we report the standard metrics Span, Nuclearity, Relation and Full F1 scores, computed using the standard Parseval [10, 117] and RST-Parseval [78] metrics.

5.4.1 Document Parsing with Gold Segmentation

Settings. Discourse parsing with gold EDUs has been the standard practice in many previous works. We compare our model with ten different baselines as shown in Table 5.1. We report most results from Zhang et al. [31], Kobayashi et al. [32], Morey et al. [117], while we reproduce Yu et al. [30] using their code.

For our model setup, we use the encoder-decoder framework with a 3-layer Bi-LSTM encoder and 3-layer unidirectional LSTM decoder. The LSTM hidden size is 400, the word embedding size is 100 for random initialization, while the character embedding size is 50. The hidden dimension in MLP modules and biaffine function for structure prediction is 500. The beam width B is 20. Our model is trained by Adam optimizer [91] with a batch size of 10000 tokens. Our learning rate is initialized at 0.002 and scheduled to decay at an exponential rate of 0.75 for every 5000 steps. Model selection for testing is performed based on the Full F1 score on the development set. When using pretrained word embeddings, we

Systems	Span	Nuc	Rel	Full
Parseval Metric [10]				
Human Agreement	78.7	66.8	57.1	55.0
Ji and Eisenstein [29] ⁺	64.1	54.2	46.8	46.3
Feng and Hirst [76] ⁺	68.6	55.9	45.8	44.6
Joty et al. [7] ⁺	65.1	55.5	45.1	44.3
Li et al. [119] ⁺	64.5	54.0	38.1	36.6
Braud et al. [109]	59.5	47.2	34.7	34.3
Braud et al. [106] [*]	62.7	54.5	45.5	45.1
Yu et al. [30] ⁺ [§]	71.4	60.3	49.2	48.1
Zhang et al. [31] ⁺	67.2	55.5	45.3	44.3
Our with GloVe	71.1	59.6	47.7	46.8
Our with XLNet [§]	74.3	64.3	51.6	50.2
RST-Parseval Metric [78]				
Human Agreement	88.7	77.3	65.4	
Yu et al. [30] ⁺ [§]	85.5	73.1	60.2	
Wang et al. [107] ⁺ [§]	86.0	72.4	59.7	
Kobayashi et al. [32] ⁺ [§]	87.0	74.6	60.0	
Our with XLNet [§]	87.6	76.0	61.8	

TABLE 5.1: Parsing results with gold segmentation. The sign ⁺ denotes that systems use handcrafted features such as lexical, syntactic, sentence/paragraph boundary features and so on, ^{*} denotes that systems use external cross-lingual features and [§] means that systems use pretrained models.

use the 100D vectors from GloVe [113]. For pretrained model, we use the *XLNet-base-cased* version [45].² The pretrained models/embeddings are kept frozen during training.

Results. From the results in Table 5.1, we see that our model with GloVe (static) embeddings achieves a Full F1 score of 46.8, the highest among all the parsers that do not use pretrained models (or contextual embeddings). This suggests that a BiLSTM-based parser can be competitive with effective modeling. The model also outperforms the one proposed by Zhang et al. [31], which is closest to ours in terms of modelling, by 3.9%, 4.1%, 2.4% and 2.5% absolute in Span, Nuclearity, Relation and Full, respectively. More importantly, our system achieves such results without relying on external data or features, in contrast to previous approaches. In addition, by using XLNet-base pretrained model [45], our system surpasses all existing methods (with or without pretraining) in all four metrics, achieving

²Our initial attempt with BERT did not offer significant gain as BERT is not explicitly designed to process long documents and has a limit of maximum 512 tokens.

Model	Span	Nuc	Rel	Full
Zhang et al. [31]	62.3	50.1	40.7	39.6
Our model				
with GloVe	63.8	53.0	43.1	42.1
with XLNet	68.4	59.1	47.8	46.6

TABLE 5.2: End-to-end parsing performance.

the state of the art with 2.9%, 4.0%, 2.4% and 2.1% absolute improvements. It also reduces the gap between system performance and human agreement. When evaluated with the RST-ParSeval [78] metric, our model outperforms the baselines by 0.6%, 1.4% and 1.8% in Span, Nuclearity and Relation, respectively.

5.4.2 Document End-to-end Parsing

For end-to-end parsing, we compare our method with the model proposed by Zhang et al. [31]. Their parsing model uses the EDU segmentation from Li et al. [120]. Our method, in contrast, predicts the EDUs along with the discourse tree in a unified process (§5.3.3). In terms of model setup, we use a setup identical to the experiments with gold segmentation (§5.4.1).

Table 5.2 reports the performance for document-level end-to-end parsing. Compared to Zhang et al. [31], our model with GloVe embeddings yields 1.5%, 2.9%, 2.4% and 2.5% absolute gains in Span, Nuclearity, Relation and Full F1 scores, respectively. Furthermore, the model with XLNet achieves even better performance and outperforms many models that use gold segmentation (Table 5.1).

EDU Segmentation Results. Our end-to-end parsing method gets an F1 score of 96.30 for the resulting EDUs. Our result rivals existing SoTA segmentation methods – 92.20 F1 of [120] and 95.55 F1 of [49]. This shows the efficacy of our unified framework for not only discourse parsing but also segmentation.³

³We could not compare our segmentation results with the DISRPT 2019 Shared Task [121] participants. We found few inconsistencies in the settings. First, in their “gold sentence” dataset, instead of using the gold sentence, they pre-process the text with an automatic tokenizer and sentence segmenter. Second, in the evaluation, under the same settings, they do not exclude the trivial BeginSegment label at the beginning of each sentence which we exclude in evaluating our segmentation result (following the RST standard).

Model	Span	Nuc	Rel	Full
Final model	71.1	59.6	47.7	46.8
w/o Beam search	70.1	58.1	46.8	45.8
w/o Boundary LSTM	68.5	55.5	46.1	44.7

TABLE 5.3: Ablation test of our models *with* gold EDUs. W/o Beam search indicates the full model with greedy decoding (beam width 1), while w/o Boundary LSTM is the full model with greedy decoding and no LSTM connection between EDU-boundary representations.

Model	Span	Nuc	Rel	Full
Final model (GloVe)	63.8	53.0	43.1	42.1
w/o GloVe	63.3	52.3	42.4	41.4
w/o Sentence guidance	59.2	48.8	40.7	38.9

TABLE 5.4: Ablation test of our end-to-end model. W/o GloVe is the full model with randomized word embeddings, while w/o Sentence guidance is the full model with randomized word embeddings and without sentence guidance.

5.4.3 Ablation Study

To further understand the contributions from the different components of our unified parsing framework, we perform an ablation study by removing selected components from a network trained with the best set of parameters.

With Gold Segmentation. For parsing with gold EDUs, Table 5.3 shows that beam search and boundary LSTM (§5.3.3) are important to the model. The former can find better tree structure by searching a larger searching space. The latter, meanwhile, connects the EDU-boundary representations, which enhances the model’s ability to capture long-range dependencies.

End-to-end Parsing. For end-to-end parsing, Table 5.4 shows that the sentence boundary constraint (§5.3.3) is indeed quite important to guide the model as it decodes long texts. Moreover, pretraining (GloVe) leads to better performance.

Error Analysis. We show our best parser’s (with gold EDUs) confusion matrix for all 18 relation labels in Fig. 5.5. The imbalanced relation distribution in RST-DT affects our model’s performance to some extent. Also semantic similar relations tend to be confused

FIGURE 5.5: Confusion matrix for relation labels on the RST-DT test set. The vertical axis represents true and horizontal axis represents predicted relations. The relations are: Joint (JO), Elaboration (EL), Temporal (TEM), Contrast (CONT), Cause (CA), Same-Unit (SA), Manner-Means (MA), Attribution (AT), Condition (COND), Comparison (COMP), Background (BA), Enablement (EN), Explanation (EX), Evaluation (EV), Summary (SU), Topic-Comment (T-CM),

with each other. Fig. 5.6 shows an example where our model mistakenly labels Summary as Elaboration. However, one could argue that the relation Elaboration is also valid here because the parenthesized text brings additional information (the equivalent amount of money). We also show more error examples for other relations (Fig. 5.7 - 5.9), where our parser labels a Condition as Background, Temporal as Joint and Explanation as Elaboration. As we can see, all these relations are semantically close and arguably interchangeable.

FIGURE 5.6: An error example where our system incorrectly labels a Summary as Elaboration.

5.4.4 Sentence Parsing

Results Table 5.5 compares the results on the discourse parsing tasks in two settings: (i) when the EDUs are given (gold segmentation) and (ii) end-to-end parsing. We see that

FIGURE 5.7: Our system incorrectly labels a Condition as Background.

FIGURE 5.8: Our system incorrectly labels a Temporal as Joint.

FIGURE 5.9: Our system incorrectly labels a Explanation as Elaboration.

our model outperforms the baselines in both parsing conditions achieving SoTA. When gold segmentation is provided, our model outperforms the single-task training model of [49] by 0.43%, 1.06% and 0.82% absolute in Span, Nuclearity and Relation, respectively. Our parser also surpasses their joint training model, which uses multi-task training (segmentation and parsing), with 0.61% and 0.4% absolute improvements in Nuclearity and Relation, respectively. For end-to-end parsing, compared to the best baseline [49], our model yields 0.27%, 0.67%, and 1.30% absolute improvements in Span, Nuclearity, Relation, respectively. This demonstrates the effectiveness of our conditional splitting approach and end-to-end formulation of the discourse analysis task. The fact that our model improves on *span* identification indicates that our method also yields better EDU segmentation.

5.4.5 Parsing Speed

Tables 5.6 and 5.7 compare the parsing speed of our models with a representative non-neural and neural models at sentence and document levels. We measure speed empirically using the wall time for parsing the test set. We ran the baselines and our models under the same settings (CPU: Intel Xeon W-2133 and GPU: Nvidia GTX 1080 Ti).

At sentence level, we follow the same set up as Lin et al. [49]. We include the model loading time for all systems. Because of using a number of features, SPADE and CODRA are typically slower than the neural models which use pretrained embeddings. As shown,

Approach	Span	Nuclearity	Relation
Parsing with gold EDU segmentation			
Human Agreement	95.7	90.4	83.0
Baselines			
Wang et al. [73]	95.6	87.8	77.6
Lin et al. [49] (single)	96.94	90.89	81.28
Lin et al. [49] (joint)	97.44	91.34	81.70
Our Model	97.37	91.95	82.10
End-to-End parsing			
Baselines			
Soricut and Marcu [69]	76.7	70.2	58.0
Joty et al. [110]	82.4	76.6	67.5
Lin et al. [49] (pipeline)	91.14	85.80	76.94
Lin et al. [49] (joint)	91.75	86.38	77.52
Our Model	92.02	87.05	78.82

TABLE 5.5: Results on sentence-level discourse parsing tasks on the RST-DT test set with and without gold segmentation.

System	Speed (Sents/s)	Speedup
End-to-End Discourse parsing (Segmenter + Parser)		
CODRA [7]	3.05	1.0x
SPADE [69]	4.90	1.6x
[49]	28.96	9.5x
Our end-to-end parser (CPU)	59.03	19.4x
Our end-to-end parser (GPU)	135.85	44.5x

TABLE 5.6: Speed comparison of our parser with existing parsers at sentence level.

our parser is 4.7x faster than the fastest end-to-end parser of Lin et al. [49], making it not only effective but also highly efficient. Running on CPU or GPU, our model is faster than all the other models, thanks to the end-to-end formulation that does not need EDU segmentation beforehand.

At document level, with gold-segmentation, our model with GloVe embeddings can parse the test set in 19 seconds, which is up to 11 times faster than [76], and this is when their features are precomputed. The speed gain can be attributed to (i) to the efficient GPU implementation of neural modules to process the decoding steps, and (ii) the fact that our model does not need to compute any handcrafted features. With pretrained models, our parser with gold segmentation is about 2.4 times faster than [30]. Our end-to-end parser that also performs segmentation is faster than the baselines that are provided with the segmentation. Nonetheless, we believe there is still room for speed improvement by choosing a better network, like the Longformer [122] which has an $\mathcal{O}(1)$ parallel time complexity in encoding a text, compared to the $\mathcal{O}(n)$ complexity of the recurrent encoder.

System	Gold Seg.	End-to-End	Time (s)	Speedup
Feng and Hirst [76]	✓		210	1.0x
Yu et al. [30]	✓		79	2.7x
Our parser (Glove)	✓		19	11.1x
Our parser (XLNet)	✓		33	6.4x
Our parser (GloVe)		✓	45	4.7x
Our parser (XLNet)		✓	60	3.5x

TABLE 5.7: The wall time for parsing the RST-DT test set at document level.

5.5 Summary

We have presented a novel top-down end-to-end method for discourse parsing based on a seq2seq model. Our model casts discourse parsing as a series of splitting decisions at token boundaries, which can solve discourse parsing and segmentation in a single model. In both end-to-end parsing and parsing with gold segmentation, our parser achieves state-of-the-art, surpassing existing methods by a good margin, without relying on handcrafted features. Our parser is not only more effective but also more efficient than the existing ones.

This work leads us to several future directions. Our short-term goal is to improve the model with better architecture and mechanisms. We also plan to explore how our method could be extrapolated to new genres like conversational or evaluative texts. We may need to adjust our model to adapt to these new domains. Moreover, speed could be further boosted by splitting the spans/nodes at the same tree level in parallel at each decoding step. We leave these extensions to future work.

Chapter 6

Differentiable Window for Dynamic Local Attention

Transformer network [3, 123] has obtained impressive results in many natural language processing tasks, thanks to its ability to capture long-range dependencies and to extract better features. It achieves this by employing efficient attention layers in parts of the network. Nonetheless, such attentions are applied globally over the key sequence. On one hand, this enables the network to extract global context easily, but on another hand does not impose many constraints if local context information is required.

In Chapters 3, 4 and 5, our proposed methods aim to learn explicit structured representations of language at the sentence and discourse level with the supervision on labeled data. We not only make the better models for hierarchical structure but adapt the tasks to be more NLU-friendly. All the proposed models do not require complex inference algorithms that are hard to incorporate into the current NLU architecture. Now all the parsing methods have simple training or inference procedures and use the cross entropy loss which is similar to NLU mainstream tasks. This may allow us to incorporate hierarchical structure to the current NLU architecture. However, learning the explicit representations and applying them to any downstream tasks require human efforts and expenses. Instead, we can still design and put some restrictions on the network to make it aware of structured or local information with least human guidance.

In this chapter, we study the strategies for learning the shallow hierarchical structure (*i.e.*, span) and incorporating local signals into attention-based neural network without

any guidance of explicit representations. This would be the first step to build the models that not only capture the context but also consider the syntax or hierarchical structure to encode the text representation. We present Differentiable Window, a new neural module and general purpose component for dynamic window selection, to enhance the ability of capturing useful local information. While universally applicable, we demonstrate a compelling use case of utilizing Differentiable Window to improve standard attention modules by enabling more focused attentions over the input regions. We propose two variants of Differentiable Window, and integrate them within the Transformer architecture in two novel ways. We evaluate our proposed approach on a myriad of NLP tasks, including machine translation, sentiment analysis, subject-verb agreement and language modeling. Empirical evaluations demonstrate consistent and sizable improvements across all tasks.¹

6.1 Introduction

Computing relative importance across a series of inputs can be regarded as one of the important advances in modern deep learning research. This paradigm, commonly known as *attention* [33], has demonstrated immense success across a wide spectrum of applications. To this end, learning to compute contextual representations [3], to point to the relevant part in the input [34], or to select windows or spans [35] from sequences forms the crux of many modern deep neural architectures.

Despite aggressive advances in developing neural modules for computing relative relevance [2, 36], there has been no general purpose solution for learning differentiable attention windows. While span selection-based pointer network models typically predict a start boundary and an end boundary [35, 37], these soft predictions generally reside at the last layer of the network and are softly optimized. To the best of our knowledge, there exists no general purpose component for learning differentiable windows within networks.

Although the practical advantages of learning differentiable windows are plenty, this chapter focuses on improving attentions with differentiable windows. The key idea is to enable more *focused* attention, leveraging dynamic window selection for limiting (and

¹This chapter is published as *Differentiable Window for Dynamic Local Attention*. *Proceedings of ACL 2020* [124].

guiding) the search space for the standard attention modules to work within. This can also be interpreted as performing a form of dynamic local attention.

We make several key technical contributions. First, we formulate the dynamic window selection problem as a problem of learning a *discrete* mask (i.e., binary values representing the window). By learning and composing left and right boundaries, we show that we are able to parameterize the (discrete) masking method. We then propose *soft* adaptations of the above mentioned, namely trainable soft masking and segment-based soft masking, which are differentiable approximations that can not only be easily optimized in an end-to-end fashion, but also inherit the desirable properties of discrete masking.

While these modules are task and model agnostic, we imbue the state-of-the-art Transformer [3] model with our differentiable window-based attention. To this end, we propose two further variants, i.e., multiplicative window attention and additive window attention for improving the Transformer model. Within the context of sequence transduction and self-attention based encoding, learning dynamic attention windows are beneficial because they can potentially eliminate noisy aggregation and alignment from large input sequences. On the other hand, it is good to note that hard attention [125], which replaces the weight average of soft attention with a stochastic sampling model, tries to achieve similar ends, albeit restricted to token-level selection. Hence, our proposed differentiable windows are more flexible and expressive compared to hard attentions.

We evaluate our Transformer model with differentiable window-based attention on a potpourri of NLP tasks, namely *machine translation*, *sentiment analysis*, *language modeling*, and *subject-verb agreement*. Extensive experimental results on these tasks demonstrate the effectiveness of our proposed method. Notably, on the English-German and English-French WMT’14 translation tasks, our method accomplishes improvements of 0.63 and 0.85 BLEU, respectively. On the Stanford Sentiment Treebank and IMDB sentiment analysis tasks, our approach achieves 2.4% and 3.37% improvements in accuracy, respectively. We further report improvements of 0.92% in accuracy and 2.13 points in perplexity on the subject-verb agreement and language modeling tasks, respectively.

6.2 Related Work

The *attention mechanism* enables dynamic selection of relevant contextual representations with respect to a query representation. It has become a key module in most deep learning models for language and image processing tasks, especially in encoder-decoder models [2, 33, 126].

6.2.1 Transformer and Global Attention

The Transformer network [3] models the encoding and decoding processes using stacked self-attentions and cross-attention (encoder-decoder attentions). Each attention layer uses a scaled multiplicative formulation defined as:

$$\text{score}(\mathbf{Q}, \mathbf{K}) = \frac{(\mathbf{Q}\mathbf{W}^Q)(\mathbf{K}\mathbf{W}^K)^T}{\sqrt{d}} \quad (6.1)$$

$$\text{att}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\text{score}(\mathbf{Q}, \mathbf{K}))(\mathbf{V}\mathbf{W}^V) \quad (6.2)$$

where $\text{softmax}(\mathbf{A})$ denotes the *softmax* operation over each row of matrix $\mathbf{A}$, $\mathbf{Q} \in \mathbb{R}^{n_q \times d}$ is the matrix containing the n_q query vectors, and $\mathbf{K}, \mathbf{V} \in \mathbb{R}^{n \times d}$ are the matrices containing the n key and value vectors respectively, with d being the number of vector dimensions; $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V \in \mathbb{R}^{d \times d}$ are the associated weights to perform linear transformations.

To encode a source sequence, the encoder applies *self-attention*, where $\mathbf{Q}, \mathbf{K}$ and $\mathbf{V}$ contain the same vectors coming from the output of the previous layer.² In the decoder, each layer first applies *masked self-attention* over previous-layer states. The resulting vectors are then used as queries to compute *cross-attentions* over the encoder states. For cross-attention, $\mathbf{Q}$ comprises the decoder self-attention states while $\mathbf{K}$ and $\mathbf{V}$ contain the encoder states. The attention mechanism adopted in the Transformer is considered **global** since the attention context spans the entire sequence.

²Initially, $\mathbf{Q}, \mathbf{K}$, and $\mathbf{V}$ contain the token embeddings.

6.2.2 Windows in Attentions

In theory, given enough training data, global attention should be able to model dependencies between the query and the key vectors well. However, in practice we have access to only a limited amount of training data. Several recent studies suggest that incorporating more focused attention over important local regions in the input sequence as an explicit inductive bias could be more beneficial.

In particular, Shaw et al. (2018) show that adding relative positional biases to the attention scores (Eq. 6.1) increases BLEU scores in machine translation. Specifically, for each query $q_i \in Q$ at position i and key $k_j \in K$ at position j , a trainable vector $a_{i,j} = w_{\max(-\tau, \min(j-i, \tau))}$ is added to the key vector before the query-key dot product is performed. The window size τ is chosen via tuning. Sperber et al. (2018) also consider local information by restricting self-attention to neighboring representations to improve long-sequence acoustic modeling. Although shown to be effective, their methods only apply to self-attention and not to cross-attention where the query vectors come from a different sequence.

That said, Luong et al. (2015) are the first to propose a Gaussian-based local attention for *cross-attention*. At each decoding step t , their model approximates the source-side pivot position p_t as a function of the decoding state and the source sequence length. Then, local attention is achieved by multiplying the attention score with a *confidence* term derived from a $\mathcal{N}(p_t, \sigma^2)$ distribution. The aligned pivot p_t and the variance σ^2 (a hyperparameter) respectively represent the center and the size of the local window.

Meanwhile, Yang et al. (2018) improve the method of Luong et al. (2015) by assigning a soft window weight (a Gaussian bias) to obtain a flexible window span. Despite effective, the aligned pivot position in the source is determined only by the decoder state, while the encoder states are disregarded - these should arguably give more relevant information regarding the attention spans over the source sequence. Besides, the confidence for local attention span may not strictly follow a normal distribution, but rather vary dynamically depending on the relationship between the query and the key. Furthermore, the approach of Luong et al. (2015) is only applicable to cross-attention while the one of Yang et al. (2018) works better only for *encoder* self-attention as shown in their experiments.

Our proposed *differentiable window* approach to local attention addresses the above limitations of previous methods. Specifically, our methods are dynamic and applicable to encoder and decoder self-attentions as well as cross-attention, without any functional constraints. They incorporate encoder states into the local window derivation. They are also invariant to sequence length, which removes the dependence on *global* features from the local context extraction process.

6.3 Dynamic Differential Window

This section outlines the overall architecture of our proposed models. The key motivation behind our architecture is to find the regions in the key-sequence for the query to focus on. Our proposed attention method works in two steps: (1) deriving the attention span for each query vector to attend over in Section 6.3 and then (2) computing the respective attention vector using the derived span in Section 6.4. In Section 6.5, we evaluate our Transformer model with differentiable window-based attention on a variety of NLP tasks, namely *machine translation*, *sentiment analysis*, *language modeling*, and *subject-verb agreement*. Extensive experimental results on these tasks demonstrate the effectiveness of our proposed method. In this section, we propose our approaches to find the attention window: *trainable soft masking* and *segment-based soft masking*. To give the necessary background to understand what can be expected from our method, we first present the *discrete masking* case.

6.3.1 Discrete Window Masking

In this context, we seek to dynamically derive a *boolean mask* vector for each query that will indicate the window in the key-sequence over which the query should attend. In other words, attentions are only activated on the consecutive positions where the mask vector element is 1, and the positions with 0 are canceled out. Let the query vector and the key-sequence be $\mathbf{q} \in \mathbb{R}^d$ and $\mathbf{K} = (\mathbf{k}_1, \mathbf{k}_2, \dots, \mathbf{k}_n)$, respectively. Formally, we define the local attention mask vector $\mathbf{m}_q \in \{0, 1\}^n$ for the query $\mathbf{q}$ as follows.

FIGURE 6.1: Example of ϕ , $\mathbf{f}$, and $\mathbf{g}$ vectors and how the mask vector $\mathbf{m}_q$ can be derived for $l_q = 3$ and $r_q = 8$.

$$\mathbf{m}_q^i = \begin{cases} 1, & \text{if } l_q \leq i \leq r_q \\ 0, & \text{otherwise} \end{cases} \quad (6.3)$$

where l_q and r_q denote the left and right positional indices that form a *discrete* window $[l_q, r_q]$ over which the query attends. As such, in the standard *global* attention, $l_q = 1$ and $r_q = n$ for all the query vectors, and in decoder self-attention, $l_q = 1$ and $r_q = t$ for the query vector at decoding step t . To facilitate the construction of $\mathbf{m}_q$, we first define vectors ϕ_k , $\mathbf{f}_k$, $\mathbf{g}_k$ and matrix $\mathbf{L}_n$ with entries as:

$$\begin{aligned} \phi_k^i &= \begin{cases} 1, & \text{if } i = k \\ 0, & \text{otherwise} \end{cases} ; \quad \mathbf{f}_k^i = \begin{cases} 1, & \text{if } i \geq k \\ 0, & \text{otherwise} \end{cases} \\ \mathbf{g}_k^i &= \begin{cases} 1, & \text{if } i \leq k \\ 0, & \text{otherwise} \end{cases} ; \quad \mathbf{L}_n^{i,j} = \begin{cases} 1, & \text{if } i \leq j \\ 0, & \text{otherwise} \end{cases} \end{aligned} \quad (6.4)$$

where $\phi_k \in \{0, 1\}^n$ denotes the one-hot representation for a boundary position k (from the left or right of a sequence), and $\mathbf{f}_k, \mathbf{g}_k \in \{0, 1\}^n$ are the ‘rightward’ mask vector and ‘leftward’ mask vector, respectively; $\mathbf{L}_n \in \{0, 1\}^{n \times n}$ denotes a unit-value (1) upper-triangular matrix with i and j being the row and column indices respectively. Figure 6.1 visualizes how these entities appear. Specifically, $\mathbf{f}_k$ has entry values of 1’s for position k and its right positions, while $\mathbf{g}_k$ has entry values of 1’s for position k and its left positions. As such, $\mathbf{f}_k$ and $\mathbf{g}_k$ can be derived from ϕ_k and $\mathbf{L}_n$ as follows.

$$\mathbf{f}_k = \phi_k^T \mathbf{L}_n; \quad \mathbf{g}_k = \phi_k^T \mathbf{L}_n^T \quad (6.5)$$

Note that $\mathbf{f}_k$ can be interpreted as the *cumulative sum* across ϕ_k , while $\mathbf{g}_k$ as the *inverse cumulative sum* across ϕ_k .

Given the above definitions, the mask vector $\mathbf{m}_q$ for a query $\mathbf{q}$ to attend over the window $[l_q, r_q]$ in the key sequence such that $1 \leq l_q \leq r_q \leq n$ can be achieved by:

$$\mathbf{m}_q = \mathbf{f}_{l_q} \odot \mathbf{g}_{r_q} = (\phi_{l_q}^T \mathbf{L}_n) \odot (\phi_{r_q}^T \mathbf{L}_n^T) \quad (6.6)$$

where $\odot$ denotes element-wise multiplication. As shown in Figure 6.1, $\mathbf{m}_q$ represents the intersection between $\mathbf{f}_{l_q}$ and $\mathbf{g}_{r_q}$, and forms a masking span for the attention.

6.3.2 Trainable Soft Masking

The above masking method is non-differentiable as ϕ is discrete, which makes it unsuitable in an end-to-end neural architecture. In our trainable soft masking method, we approximate the discrete one-hot vector ϕ with a pointing mechanism [34].³ Specifically, given the query $\mathbf{q}$ and the key-sequence $\mathbf{K}$ as before, we define confidence vectors $\hat{\phi}_{l_q}, \hat{\phi}_{r_q} \in \mathbb{R}^n$ as follows.

$$\hat{\phi}_{l_q} = \text{softmax}\left(\frac{\mathbf{q}^T \mathbf{W}_L^Q (\mathbf{K} \mathbf{W}_L^K)^T}{\sqrt{d}}\right) \quad (6.7)$$

$$\hat{\phi}_{r_q} = \text{softmax}\left(\frac{\mathbf{q}^T \mathbf{W}_R^Q (\mathbf{K} \mathbf{W}_R^K)^T}{\sqrt{d}}\right) \quad (6.8)$$

where softmax is the *softmax* function as defined before, and $\mathbf{W}_L^Q, \mathbf{W}_L^K, \mathbf{W}_R^Q, \mathbf{W}_R^K \in \mathbb{R}^{d \times d}$ are trainable parameters. Eq. 6.7-6.8 approximate the left and right boundary positions of the mask vector for the query $\mathbf{q}$. However, contrary to the discrete case, they do not enforce *absolute* cancellation or activation of attention weights on any position in the key-sequence. Instead, they assign a confidence score to each position. This allows the

³However, unlike the standard pointer network, in our case there is no direct supervision for learning the pointing function. Our network instead learns it from the end prediction task.

model to gradually correct itself from invalid assignments. Moreover, the *softmax* operations enable differentiability while maintaining the gradient flow in an end-to-end neural architecture.

Note however that the left and right boundary concepts have now become ambiguous since the positions $l_q = \arg \max(\hat{\phi}_{l_q})$ and $r_q = \arg \max(\hat{\phi}_{r_q})$ are not guaranteed to conform to the constraint $l_q \leq r_q$. To understand its implication, let's first consider the discrete case in Eq. 6.6; the element-wise multiplication between $\mathbf{f}_{l_q}$ and $\mathbf{g}_{r_q}$ results in a *zero* vector for $\mathbf{m}_q$ if $l_q > r_q$, canceling out the attention scores entirely. Although not absolute *zeros*, in the continuous case, $\mathbf{m}_q$ would potentially contain significantly small values, which renders the attention implausible. To address this, we compute the *soft* mask vector $\hat{\mathbf{m}}_q$ as follows.

$$\hat{\mathbf{m}}_q = (\hat{\phi}_{l_q}^T \mathbf{L}_n) \odot (\hat{\phi}_{r_q}^T \mathbf{L}_n) + (\hat{\phi}_{r_q}^T \mathbf{L}_n) \odot (\hat{\phi}_{l_q}^T \mathbf{L}_n) \quad (6.9)$$

This formulation has two additive terms; the former constructs the mask vector when $l_q \leq r_q$, whereas the latter is activated when $l_q > r_q$. This ensures a non-zero result regardless of l_q and r_q values. It can be shown that the values in $\hat{\mathbf{m}}_q$ represent the expected value of the discrete flags in $\mathbf{m}_q$, i.e., $\hat{\mathbf{m}}_q = \mathbb{E}(\mathbf{m}_q)$; see Appendix for a proof.

We concatenate the mask vectors horizontally for all the query vectors in $\mathbf{Q} \in \mathbb{R}^{m \times d}$ to get the mask matrix $\mathbf{M} \in \mathbb{R}^{m \times n}$. Since the pointing mechanism is invariant to sequence length, the computation of the mask vectors enjoys the same advantages, enabling our models to efficiently perform attentions on any arbitrarily long sequences. In addition, the method is applicable to all attention scenarios – from decoder to encoder cross-attention, encoder self-attention, and decoder self-attention.

6.3.3 Segment-Based Soft Masking

The soft masking introduced above modulates the attention weight on each token separately which may result in unsmooth attention weights on neighbouring tokens. However, words in a sentence are related and they often appear in chunks or phrases, contributing to a shared meaning. Thus, it may be beneficial to assign identical mask values to the tokens within a segment so that they are equally treated in the window selection method.

In this section, we propose a novel extension to our soft masking method that enables the mask vector to share the same masking values for the tokens within a segment in a key-sequence.

The main idea is to divide the key-sequence $\mathbf{K} = (\mathbf{k}_1, \mathbf{k}_2, \dots, \mathbf{k}_n)$ into $\lceil n/b \rceil$ consecutive segments and to assign the same masking value to the tokens in a segment. The segment size b is considered a hyper-parameter. We compute the segment-based mask vector $\mathbf{m}'_q$ similarly as in Eq. 6.9, but with $\mathbf{L}_n$ replaced by $\mathbf{J}_n \in \mathbb{R}^{n \times n}$ defined as follows.

$$\mathbf{J}_n^{i,j} = \begin{cases} 1, & \text{if } i \leq b \lceil \frac{j}{b} \rceil \\ 0, & \text{otherwise} \end{cases} \quad (6.10)$$

$$\mathbf{m}'_q = (\hat{\phi}_{l_q}^T \mathbf{J}_n) \odot (\hat{\phi}_{r_q}^T \mathbf{J}_n^T) + (\hat{\phi}_{r_q}^T \mathbf{J}_n) \odot (\hat{\phi}_{l_q}^T \mathbf{J}_n^T) \quad (6.11)$$

Eq. 6.10 - 6.11 ensure that all the items in a segment share the same masking value, which is the cumulative sum of the confidence scores in $\hat{\phi}_{l_q}$ and $\hat{\phi}_{r_q}$. For instance, suppose $\hat{\phi}_{l_q} = (a_1, a_2, a_3, \dots, a_n)$ and segment size $b = 2$, then the term $\hat{\phi}_{l_q}^T \mathbf{J}_n$ evaluates to $(\sum_{i=1}^2 a_i, \sum_{i=1}^2 a_i, \sum_{i=1}^4 a_i, \dots)$, and $\hat{\phi}_{l_q}^T \mathbf{J}_n^T$ evaluates to $(\sum_{i=1}^n a_i, \sum_{i=1}^n a_i, \sum_{i=3}^n a_i, \dots)$. Similarly, $\hat{\phi}_{r_q}^T \mathbf{J}_n^T$ and $\hat{\phi}_{r_q}^T \mathbf{J}_n$ will have segment-level effects on the cumulative sums. Figure 6.2 visualizes the method with an example for $b = 2$.

One advantage of this approach is that it allows us to control the masking behavior (by varying b) without increasing the number of parameters compared to the token-based masking. We also show its effectiveness in our experiments.

6.4 Dynamic Window Attention Methods

Having presented our method to compute the mask vector that defines the attention spans, we now present our methods to incorporate the mask vectors into the attention layers.

FIGURE 6.2: Segment-based masking for segment size = 2. Instead of pointing to the left and right indices of the *tokens*, the soft segment-based method (approximately) points to the left and right boundaries of the *segments*, respectively.

6.4.1 Multiplicative Window Attention

In this approach, the attention weights (Eq. 6.2) are (element-wise) multiplied by the mask matrix M to confine their attention scope defined by the mask. Formally, the attention scores and outputs are defined as follows.

$$\text{score} = \frac{(QW^Q)(KW^K)^T}{\sqrt{d}} \quad (6.12)$$

$$\text{att}_{\text{MW}} = (\text{softmax}(\text{score}) \odot M)(VW^V) \quad (6.13)$$

In this approach, the standard global attention weights are suppressed and partially overshadowed by the attention window imposed by M . Thus, it can be interpreted as a *local attention* method similar to Luong et al. (2015). However, instead of using a static Gaussian bias, we use a dynamic mask to modulate the attention weights.

6.4.2 Additive Window Attention

Having a *local* attention window could be beneficial, but it does not rule out the necessity of global attention, which has been shown effective in many applications [3, 44]. Thus, we also propose an *additive window attention*, which implements a combination of global attention and local attention. The attention output in this method is formally defined as

$$s_{\text{glb}} = (QW_{\text{glb}}^Q)(QW_{\text{glb}}^K)^T \quad (6.14)$$

$$s_{\text{loc}} = (QW_{\text{loc}}^Q)(QW_{\text{loc}}^K)^T \odot M \quad (6.15)$$

$$\text{score}_{\text{AW}} = \frac{s_{\text{glb}} + s_{\text{loc}}}{\sqrt{d}} \quad (6.16)$$

$$\text{att}_{\text{AW}} = \text{softmax}(\text{score}_{\text{AW}})(VW^V) \quad (6.17)$$

where W_{glb}^Q , W_{glb}^K , W_{loc}^Q , and $W_{\text{loc}}^K \in \mathbb{R}^{d \times d}$ are the weight matrices for global and local attentions.

Compared to the multiplicative window attention where the mask re-evaluates the global attention weights, additive window attention applies the mask vector to the *local attention scores* (s_{loc}), which is then added to the *global attention scores* (s_{glb}) before passing it through the softmax function. In this way, the mask-defined local window does not suppress the global context but rather complements it with a local context. Moreover, the resulting attention weights add up to *one*, which avoids attention weights diminishment that could occur in the multiplicative window attention. Additive merger of global and local window components may also facilitate more stable gradient flows.

6.4.3 Implementation in Transformer

We now describe how the proposed dynamic window attention methods can be integrated into the Transformer.

Encoder, Decoder and Cross Attentions. Our proposed methods can be readily applied to the any of the attention layers in the Transformer framework. We could also selectively apply our methods to different layers in the encoder and decoder. In our initial experiments on WMT’14 English-German development set, we observed that the following settings provide more promising performance gains.

First, encoder self-attention layers benefit most from *additive window attention*, while decoder self-attention layers prefer *multiplicative attention*. This shows that the *global* attention component is more useful when the key sequence is provided entirely in the encoder,

while less useful when only the fragmented key sequence (past keys) is visible in the decoder. Second, the above argument is further reinforced as we found that cross-attention layers also prefer *additive window attention*, where the entire source sequence is available. Third, cross-attention works better with *segment-based masking*, which provides smoothness and facilitates phrase (n-gram) based translations.

Lower-layer Local Attentions. It has been shown that deep neural models learn simple word features and local syntax in the lower layers, while higher layers learn more complex context-dependent aspects of word semantics. Belinkov et al. (2017) show this on NMT models, while Peters et al. (2018) and Jawahar et al. (2019) show this on representation learning with ELMo and BERT respectively. In other words, local contextual information can still be derived in higher layers with the standard global attention. As such, we propose to apply our dynamic window attention methods only to the first 3 layers of the Transformer network, leaving the top 3 layers intact. Our diverse experiments in the following section support this setup as it offers substantial improvements, whereas using local attention in higher layers does not show gains, but rather increases model parameters.

6.5 Experiment

In this section, we present the training settings, experimental results and analysis of our models in comparison with the baselines on machine translation (MT), sentiment analysis, subject verb agreement and language modeling (LM) tasks.

6.5.1 Machine Translation

We trained our models on the standard WMT’16 English-German (En-De) and WMT’14 English-French (En-Fr) datasets containing about 4.5 and 36 million sentence pairs, respectively. For validation (development) purposes, we used *newstest2013* for En-De and a random split from the training set for En-Fr. All translation tasks were evaluated against their respective *newstest2014* test sets, in case-sensitive tokenized BLEU. We used *byte-pair encoding* [132] with shared source-target vocabularies of 32,768 and 40,000 sub-words for

En-De and En-Fr translation tasks, respectively. We compare our models with three strong baselines: (i) Transformer Base [3], (ii) Transformer Base with Relative Position [42], and (ii) Transformer Base with Localness Modeling [128]. To ensure a fair comparison, we trained our models and the baselines with the following training setup.

Training Setup. We followed model specifications in [3] and optimization settings in [133], with some minor modifications. Specifically, we used word embeddings of dimension 512, feedforward layers with inner dimension 2048, and multi-headed attentions with 8 heads.

We trained our models on a single physical GPU but replicated the 8-GPU setup following the *gradient aggregation* method proposed by Ott et al. (2018).

We trained the models for 200,000 *updates* for En-De and 150,000 *updates* for En-Fr translation tasks. Finally, we averaged the last 5 checkpoints to obtain the final models for evaluation. The segment size b in the segment-based masking method was set to 5.⁴

Translation Results. We report our translation results in Table 6.1; **Enc(AW)** indicates the use of *additive window* (AW) attention in the encoder, **Dec(MW)** indicates the use of *multiplicative window* (MW) attention in the decoder, and **Cr(AW,Seg)** indicates the use of additive window attention with *segment-based masking* for cross-attention. The attention module that is not specified in our naming convention uses the default token-based global attention in the Transformer. For example, **Enc(AW)-Dec(MW)** refers to the model that uses AW attention in the encoder, MW attention in the decoder and the default global attention for cross attention.

We notice that despite a minor increase in the number of parameters, applying our attentions in the encoder and decoder offers about 0.7 and 1.0 BLEU improvements in En-De and En-Fr translation tasks respectively, compared to the Transformer base [3]. Our model with the segment-based additive method for *cross* attention achieves a similar performance. We observe further improvements as we apply our attentions in all the attention modules of the Transformer. Specifically, our model Enc(AW)-Cr(AW,Seg)-Dec(MW)

⁴We did not tune b ; tuning b might improve the results further.

achieves 28.25 and 40.32 BLEU in En-De and En-Fr translation tasks, outperforming Transformer base with localness [128] by 0.63 and 0.85 BLEU, respectively.

Model	#-params	En-De	En-Fr
Vaswani et al. [3]	63M	27.46	39.21
Shaw et al. [42]	63M	27.56	39.37
Yang et al. [128]	63M	27.62	39.47
Our Models			
Enc(AW)-Dec(MW)	68M	28.11	40.24
Cr(AW, Seg)	65M	28.13	40.06
Enc(AW)-Cr(AW,Seg)-Dec(MW)	73M	28.25	40.32

TABLE 6.1: BLEU scores for different models in WMT’14 English-German and English-French translation tasks.

6.5.2 Ablation Study

To verify our modeling decisions, we performed an ablation study in the WMT’14 En-De translation task. In particular, we evaluated (i) the impact of applying our differentiable window attentions in all layers vs. only in certain lower layers of the Transformer network, (ii) which window attention methods (additive or multiplicative) are suitable particularly for the encoder/decoder self-attention and cross-attention, and (iii) the impact of segment-based masking in different attention modules. (iv) training efficiency and performance of our best model with the similar models. Plus, to further interpret our window-based attention, we also provide the local window visualization.

Method	Module	Full (6 layers)	Partial (3 layers)
Transformer	-	27.46	-
AW	Encoder	27.77	27.90
MW	Encoder	27.25	27.40
AW	Decoder	27.73	27.85
MW	Decoder	27.88	28.04
AW	Cross	27.78	27.97
MW	Cross	27.58	27.79

TABLE 6.2: Evaluation of Additive Window (AW) and Multiplicative Window (MW) attentions in encoder/decoder self attention and cross attention for full vs. partial settings.

Full vs. Partial. Table 6.2 shows BLEU scores for the Transformer models that employ our window-based attentions in all 6 layers (**Full**) vs. only in the first 3 layers (**Partial**), as well as the methods used in different attention modules (encoder/decoder self-attention, cross-attention). We can see that almost all the models with window-based methods in the first 3 layers outperform those that use them in all 6 layers. This gives the setup significant advantages as it performs not only better in BLEU but also requires less parameters.

The results also show that multiplicative window (MW) attention is preferred in decoder self-attention, while additive window (AW) is more suitable for encoder self-attention and for cross-attention. This suggests that the global context, which is maintained in AW, is more useful when it is entirely available like in encoder self-attention and cross attention. In contrast, incomplete and partially-generated context in decoder self-attention may induce more noise than information, where MW attention renders better performance than AW.

Model	Token-based	Segment-based
Cr(AW)	27.97	28.13
Enc(AW)-Dec(MW)	28.11	27.91

TABLE 6.3: BLEU scores for token- and segment-based masking in cross attention and encoder self-attention. The decoder self-attention always uses token-based masking.

Token- vs. Segment-based. Table 6.3 compares the results for using token-based vs. segment-based masking methods in different attention modules of the network. Note that it is preferred for decoder self-attention to adopt token-based masking since the decoder cannot point to *unfinished* segments in autoregressive generation, if it had used segment-based masking. We see that segment-based additive window masking outdoes its token-based counterpart (28.13 vs. 27.97 BLEU) for cross-attention. Meanwhile, for encoder self-attention, token-based masking performs better than segment-based masking by 0.2 BLEU. This suggests that segments (or phrases) represent better *translation* units than tokens, justifying its performance superiority in cross-lingual attention but not in monolingual (self-attention) encoding.

Model	#-params	# steps/sec	BLEU
Vaswani et al. [3]	63M	1.20	27.46
Yang et al. [128]	63M	1.07	27.62
Vaswani et al. [3] 7 layers	69M	1.05	27.74
Vaswani et al. [3] 8 layers	75M	0.99	27.89
Enc(AW)-Cr(AW,Seg)-Dec(MW)	73M	1.04	28.25

TABLE 6.4: Training efficiency and size of similar models

Speed and Parameters. As shown in table 6.4, our training efficiency is competitive to the baselines. That is, the training speed for our model is 1.04 steps/sec which is similar to Yang et al. [128]. Besides, our model outperforms the Transformer with 8 layers, which has more parameters. This suggests that our performance gain may not come from additional parameters, but rather from a better inductive bias through the dynamic window attention.

Local Window Visualization. To further interpret our window-based attentions, Figure 6.3(a) shows the cross-attention soft masking values ($\hat{m}_q$) on the source tokens for each target token in an En-Fr test sample assigned by our Enc(AW)-Cr(AW,Seg)-Dec(MW) model. The darker the score, the higher the attention is from a target token to a source token. We can see the relevant subwords are captured by the attentions quite well, which promotes ngram-level alignments. For instance, the mask ($\hat{m}_q$) guides the model to evenly distribute attention scores on sub-words “Co@@” and “en” (Fig. 6.3(b)), while standard attention is biased towards “Co@@” (Fig. 6.3(c)). Similar phenomenon can be seen for “Bro@@” and “thers” (towards “frères”).

FIGURE 6.3: Visualization of masking scores, and attention scores for our and the original Transformer models.

Model	STT	IMDB	SVA
Vaswani et al. [3]	79.36	83.65	94.48
Shaw et al. [42]	79.73	84.61	95.27
Yang et al. [128]	79.24	84.13	95.00
Enc (MW)	79.70	85.09	95.95
Enc (AW)	82.13	87.98	96.19

TABLE 6.5: Classification accuracy on Stanford Sentiment Treebank (SST) and IMDB sentiment analysis and Subject-Verb Agreement(SVA) tasks.

6.5.3 Text Classification

We evaluate our models on the Stanford Sentiment Treebank (SST) [134], IMDB sentiment analysis [135] and Subject-Verb Agreement (SVA) [136] tasks. We compare our attention methods (incorporated into the Transformer encoder) with the encoders of Vaswani et al. (2017), Shaw et al. (2018) and Yang et al. (2018).

Training Setup. As the datasets are quite small compared to the MT datasets, we used *tiny versions* of our models as well as the baselines.⁵ Specifically, the models consist of a 2-layer Transformer encoder with 4 attention heads, 128 hidden dimensions and 512 feedforward inner dimensions. In these experiments, our attention methods are applied only to the first layer of the network. We trained for 3,000, 10,000 and 10,000 updates for SST, IMDB and SVA tasks, respectively on a single GPU machine.

Results. Table 6.5 shows the results. Our multiplicative window approach (Enc (MW)) achieves up to 79.7%, 85.1% and 95.95% accuracy in SST, IMDB and SVA, exceeding Transformer [3] by 0.4%, 1.35% and 1.47%, respectively. Our additive window attention (Enc (AW)) renders even more improvements. Specifically, it outperforms Transformer with relative position (Shaw et al. 2018) by 2.4% and 3.37%, 0.92% reaching 82.13%, 87.98% and 96.19% accuracy in SST, IMDB and SVA, respectively. In fact, the results demonstrate consistent trends with our earlier MT experiments: additive window attention outdoes its multiplicative counterpart in the encoder, where the entire key sequence is available.

⁵As specified in <https://github.com/tensorflow/tensor2tensor>.

Model	Perplexity
Vaswani et al. [3]	46.37
Shaw et al. [42]	46.13
Dec (MW)	44.00
Dec (AW)	44.95

TABLE 6.6: Perplexity scores on 1-billion-word language modeling benchmark (the lower the better).

6.5.4 Language Modeling

Finally, to demonstrate our proposed methods as effective general purpose NLP components, we evaluate them on the One Billion Word LM Benchmark dataset [137]. The dataset contains 768 million words of data compiled from WMT 2011 News Crawl data, with a vocabulary of 32,000 words. We used its held-out data as the test set.

Training Setup. As the LM dataset is considerably large, we used the same model settings as adopted in our MT experiments. For these experiments, we only trained the models on virtually 4 GPUs for 100,000 *updates* using gradient aggregation on a single GPU machine. Note that only the self-attention based autoregressive decoder of the Transformer framework is used in this task. Therefore, the method of Yang et al. (2018) is not applicable to this task.

Results. Table 6.6 shows the perplexity scores. As can be seen, our multiplicative and additive window attention models both surpass Transformer [3] by 2.37 and 1.42 points respectively, reaching 44.00 and 44.95 perplexity scores respectively. In addition, it is noteworthy that similar to MT experiments, multiplicative attention outperforms the additive one on this task, where the decoder is used. This further reinforces the claim that where the global context is not *fully* available like in the decoder, the incomplete global context may induce noises into the model. Thus, it is effective to embrace dynamic local window attention to suppress the global context, for which the multiplicative window attention is designed.

6.6 Summary

We have presented a novel Differential Window method for dynamic window selection, and used it to improve the standard attention modules by enabling more focused attentions. Specifically, we proposed Trainable Soft Masking and Segment-based Masking, which can be applied to encoder/decoder self-attentions and cross attention.

We evaluated our models on four NLP tasks including machine translation, sentiment analysis, subject verb agreement and language modeling. Our experiments show that our proposed methods outperform the baselines significantly across all the tasks. All in all, we demonstrate the benefit of incorporating the differentiable window in the attention. In the future, we would like to extend our work to make a syntactically-aware window that can automatically learn tree (or phrase) structures.

Chapter 7

Conclusion

This chapter concludes the thesis with an overall summary of the contributions and discusses a number of future directions.

7.1 Summary

In this dissertation, we demonstrated that structured pointing network can be used to improve NLU in many aspects. The first part of the dissertation focused on developing more effective and efficient parsers. In [Chapter 3](#), we proposed a novel approach for constituency parsing that converts the tree structure into a series of span-oriented pointing decisions and separate the structure and label prediction tasks. This transformation enables efficient and structure-consistent computation of the probability of a span being a node in the tree. With an efficient top-down inference algorithm, our parser outperforms the existing parsers with similar time complexity.

The top-down idea was improved further in [Chapter 4](#), where we built a conditional split-based parser that casts the constituency tree into a sequence of splitting decisions. We utilized the seq2seq framework to consider splitting history in making current splitting decisions together with efficient beam search inference. This simple idea enables our parser to outperform most of existing parsers in not only accuracy but also speed.

In [Chapter 5](#), we extend this idea further to discourse parsing at the sentence-level and document-level. Our model can generate a discourse tree without requiring discourse

segmentation, in other words, perform both discourse parsing and discourse segmentation in a single framework.

The second part of the dissertation ([Chapter 6](#)) then concentrated on enhancing the ability to capture more relevant context of languages with more comprehensive inductive bias. With a pointing mechanism, we study the strategies for learning the shallow hierarchical structure and incorporating local signals into an attention-based neural network. We proposed a new neural module that are suitable and effective on a number of NLU tasks including machine translation, sentiment analysis, subject-verb agreement and language modeling.

7.2 Future Directions

There are still many open challenges to be addressed in applying pointing mechanism in natural language understanding. While we discussed future direction at the end of the corresponding chapters, in this section, we briefly summarise the general future directions of the work presented in this dissertation.

- **Unsupervised Constituency Parsing:** In all the constituency parsing methods proposed in this thesis, we required supervision on labeled data which costs human efforts and expenses. It would be interesting to see how effective our models could be when they are applied under unsupervised settings like [\[138\]](#). The fact that our models separate the structure and label prediction could make it easier for them to be adapted to unsupervised settings because unsupervised parsing generally only considers unlabelled structure.
- **Faster and More Efficient Parsing Methods:** A key limitation in our conditional parsers in [Chapters 4](#) and [5](#) is that they use recurrent neural networks in the encoder which can limit the speed and representation ability of the models. These parsers can still be improved with better architecture (e.g., Transformer) to achieve better accuracy and speed. Moreover, speed could be further boosted by splitting the spans/nodes at the same tree level in parallel at each decoding step as described in [Chapter 5](#).

- **A Joint Model Integrating All Parsing Tasks:** In this dissertation, we considered constituency and discourse parsing separately. However, there is a line of previous research that builds the discourse tree from constituency tree [112]. This shows that there is a strong connection between these two parsing tasks. Therefore, it is reasonable to integrate these tasks in a joint model. Our framework may also allow to do dependency parsing [68]. It is also possible to do it in multilingual settings where there are similarities between these tree structures between different languages. We also plan to explore how our method could be extrapolated to new genres like conversational or evaluative texts.
- **Incorporating Syntactic Information Into Neural Modules:** Turning to designing the specific purpose neural modules for NLU, our proposed attention method (Chapter 6) could be used to make a syntax-aware window that can automatically learn the syntactic structures.

List of Publications

The following is a list of my publications during my PhD studies.

1. Efficient Constituency Parsing by Pointing. **Thanh-Tung Nguyen**, Xuan-Phi Nguyen, Shafiq Joty, Xiaoli Li. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL 2020)*, Pages 3284-3294, Virtual.
2. Differentiable Window for Dynamic Local Attention. **Thanh-Tung Nguyen**, Xuan-Phi Nguyen, Shafiq Joty, Xiaoli Li. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL 2020)*, Pages 6589-6599, Virtual.
3. A Conditional Splitting Framework for Efficient Constituency Parsing. **Thanh-Tung Nguyen**, Xuan-Phi Nguyen, Shafiq Joty, Xiaoli Li. *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics (ACL 2021)*, Virtual.
4. RST parsing from scratch. **Thanh-Tung Nguyen**, Xuan-Phi Nguyen, Shafiq Joty, and Xiaoli Li. *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (NAACL21)*, Virtual.
5. Cross-model Back-translated Distillation for Unsupervised Machine Translation. Xuan-Phi Nguyen, Shafiq Joty, **Thanh-Tung Nguyen**, Wu Kui, and Ai Ti. *In Thirty-eighth International Conference on Machine Learning (ICML'21)*, Virtual.
6. Adaptation of hierarchical structured models for speech act recognition in asynchronous conversation. **Thanh-Tung Nguyen**, Tasnim Mohiuddin, Shafiq Joty. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (NAACL-HLT, 2019)*, pages 1326-1336, Minneapolis, Minnesota.

7. Modeling marked temporal point process using multi-relation structure RNN. Hongyun Cai, **Thanh-Tung Nguyen**, Yan Li, Vincent W Zheng, Binbin Chen, Gao Cong, Xiaoli Li. In **Cognitive Computation 2020.**, pages, 499–512.

References

- [1] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. *Dive into Deep Learning*. Virtual, 2020. <https://d2l.ai>.
- [2] Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, EMNLP, pages 1412–1421. ACL, 2015.
- [3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.
- [4] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2692–2700. Curran Associates, Inc., 2015. URL <http://papers.nips.cc/paper/5866-pointer-networks.pdf>.
- [5] Daniel Jurafsky and James H. Martin. *Speech and language processing*. Prentice Hall, Upper Saddle River, 3rd edition, 2020.
- [6] Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. Recurrent neural network grammars. In Kevin Knight, Ani Nenkova, and Owen Rambow, editors, *HLT-NAACL*, pages 199–209. The Association for Computational Linguistics, 2016.

- [7] Shafiq Joty, Giuseppe Carenini, and Raymond T. Ng. CODRA: A novel discriminative framework for rhetorical analysis. *Computational Linguistics*, 41(3):385–435, September 2015.
- [8] Jiwei Li, Rumeng Li, and Eduard Hovy. Recursive deep models for discourse parsing. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2061–2069, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1220. URL <https://www.aclweb.org/anthology/D14-1220>.
- [9] Amandla Mabona, Laura Rimell, Stephen Clark, and Andreas Vlachos. Neural generative rhetorical structure parsing. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2284–2295, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1233. URL <https://www.aclweb.org/anthology/D19-1233>.
- [10] Mathieu Morey, Philippe Muller, and Nicholas Asher. How much progress have we made on RST discourse parsing? a replication study of recent results on the RST-DT. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1319–1324, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/D17-1136. URL <https://www.aclweb.org/anthology/D17-1136>.
- [11] M. A. K. Halliday and R. Hasan. *Cohesion in English*. Longman, London, 1976.
- [12] Barbara J. Grosz and Candace L. Sidner. Attention, intentions, and the structure of discourse. *Computational Linguistics*, 12(3):175–204, July 1986. ISSN 0891-2017. URL <http://dl.acm.org/citation.cfm?id=12457.12458>.
- [13] William C. Mann and Sandra A. Thompson. Rhetorical structure theory: Toward a functional theory of text organization. *Text*, 8(3):243–281, 1988.

- [14] Barbara J. Grosz, Aravind K. Joshi, and Scott Weinstein. Centering: A framework for modeling the local coherence of discourse. *Computational Linguistics*, 21(2): 203–225, 1995.
- [15] Hans Kamp and Uwe Reyle. *From Discourse to Logic: Introduction to Model-theoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory*, volume 42 of *Studies in Linguistics and Philosophy*. Springer-Verlag, New York, 1993.
- [16] Bonnie L. Webber. D-ltag: extending lexicalized tag to discourse. *Cogn. Sci.*, 28(5): 751–779, 2004.
- [17] Rashmi Prasad, Nikhil Dinesh, Alan Lee, Eleni Miltsakaki, Livio Robaldo, Aravind K. Joshi, and Bonnie L. Webber. The penn discourse treebank 2.0. In *LREC*. European Language Resources Association, 2008.
- [18] Mengqiu Wang, Noah A. Smith, and Teruko Mitamura. What is the Jeopardy model? a quasi-synchronous grammar for QA. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 22–32, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL <https://www.aclweb.org/anthology/D07-1003>.
- [19] Aliaksei Severyn and Alessandro Moschitti. Automatic feature engineering for answer selection and extraction. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 458–467, Seattle, Washington, USA, October 2013. Association for Computational Linguistics. URL <https://www.aclweb.org/anthology/D13-1044>.
- [20] Michael Heilman and Noah A. Smith. Tree edit models for recognizing textual entailments, paraphrases, and answers to questions. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 1011–1019, Los Angeles, California, June 2010. Association for Computational Linguistics. URL <https://www.aclweb.org/anthology/N10-1145>.

- [21] Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. Recurrent neural network grammars. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 199–209, San Diego, California, June 2016. Association for Computational Linguistics. doi: 10.18653/v1/N16-1024. URL <https://www.aclweb.org/anthology/N16-1024>.
- [22] James Cross and Liang Huang. Span-based constituency parsing with a structure-label system and provably optimal dynamic oracles. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1–11, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1001. URL <https://www.aclweb.org/anthology/D16-1001>.
- [23] Jiangming Liu and Yue Zhang. Shift-reduce constituent parsing with neural look-ahead features. *Transactions of the Association for Computational Linguistics*, 5: 45–58, 2017. doi: 10.1162/tacl_a_00045. URL <https://www.aclweb.org/anthology/Q17-1004>.
- [24] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28, pages 1171–1179. Curran Associates, Inc., 2015. URL <https://proceedings.neurips.cc/paper/2015/file/e995f98d56967d946471af29d7bf99f1-Paper.pdf>.
- [25] Greg Durrett and Dan Klein. Neural CRF parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 302–312, Beijing, China, July 2015. Association for Computational Linguistics. doi: 10.3115/v1/P15-1030. URL <https://www.aclweb.org/anthology/P15-1030>.
- [26] David Gaddy, Mitchell Stern, and Dan Klein. What’s going on in neural constituency parsers? an analysis. In *Proceedings of the 2018 Conference of the North American*

- Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 999–1010. Association for Computational Linguistics, June 2018. doi: 10.18653/v1/N18-1091. URL <https://www.aclweb.org/anthology/N18-1091>.
- [27] Nikita Kitaev and Dan Klein. Constituency parsing with a self-attentive encoder. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2676–2686, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1249. URL <https://www.aclweb.org/anthology/P18-1249>.
- [28] Nikita Kitaev, Steven Cao, and Dan Klein. Multilingual constituency parsing with self-attention and pre-training. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3499–3505, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1340. URL <https://www.aclweb.org/anthology/P19-1340>.
- [29] Yangfeng Ji and Jacob Eisenstein. Representation learning for text-level discourse parsing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13–24, Baltimore, Maryland, June 2014. Association for Computational Linguistics. doi: 10.3115/v1/P14-1002. URL <https://www.aclweb.org/anthology/P14-1002>.
- [30] Nan Yu, Meishan Zhang, and Guohong Fu. Transition-based neural RST parsing with implicit syntax features. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 559–570, Santa Fe, New Mexico, USA, August 2018. Association for Computational Linguistics. URL <https://www.aclweb.org/anthology/C18-1047>.
- [31] Longyin Zhang, Yuqing Xing, Fang Kong, Peifeng Li, and Guodong Zhou. A top-down neural architecture towards text-level parsing of discourse rhetorical structure. In *Proceedings of the 58th Annual Meeting of the Association for Computational*

- Linguistics*, pages 6386–6395, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.569. URL <https://www.aclweb.org/anthology/2020.acl-main.569>.
- [32] Naoki Kobayashi, Tsutomu Hirao, Hidetaka Kamigaito, Manabu Okumura, and Masaaki Nagata. Top-down rst parsing utilizing granularity levels in documents. In *Proceedings of the 2020 Conference on Artificial Intelligence for the American (AAAI)*, pages 8099–8106, sep 2020.
- [33] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *ICLR*, 2015.
- [34] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2692–2700. Curran Associates, Inc., 2015. URL <http://papers.nips.cc/paper/5866-pointer-networks.pdf>.
- [35] Shuohang Wang and Jing Jiang. Machine comprehension using match-lstm and answer pointer. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.
- [36] Chung-Cheng Chiu and Colin Raffel. Monotonic chunkwise attention. In *International Conference on Learning Representations*, 2018. URL <https://openreview.net/pdf?id=Hko85plCW>.
- [37] Min Joon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.
- [38] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning Representations by Back-propagating Errors. *Nature*, 323(6088):533–536, 1986. doi: 10.1038/323533a0. URL <http://www.nature.com/articles/323533a0>.

- [39] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [40] Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *CoRR*, 2014. URL <http://arxiv.org/abs/1409.1259>.
- [41] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014. URL <http://arxiv.org/abs/1412.3555>.
- [42] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 464–468. Association for Computational Linguistics, 2018. doi: 10.18653/v1/N18-2074. URL <http://aclweb.org/anthology/N18-2074>.
- [43] Yi Tay, Anh Tuan Luu, and Siu Cheung Hui. Hermitian co-attention networks for text matching in asymmetrical domains. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 4425–4431. International Joint Conferences on Artificial Intelligence Organization, 7 2018. doi: 10.24963/ijcai.2018/615. URL <https://doi.org/10.24963/ijcai.2018/615>.
- [44] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL <https://www.aclweb.org/anthology/N19-1423>.
- [45] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for

- language understanding. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32, pages 5753–5763. Curran Associates, Inc., 2019. URL <https://proceedings.neurips.cc/paper/2019/file/dc6a7e655d7e5840e66733e9ee67cc69-Paper.pdf>.
- [46] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. URL <http://arxiv.org/abs/1512.03385>.
- [47] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016. URL <http://arxiv.org/abs/1607.06450>.
- [48] Xuezhe Ma, Zecong Hu, Jingzhou Liu, Nanyun Peng, Graham Neubig, and Eduard Hovy. Stack-pointer networks for dependency parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1403–1414, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1130. URL <https://www.aclweb.org/anthology/P18-1130>.
- [49] Xiang Lin, Shafiq Joty, Prathyusha Jwalapuram, and M Saiful Bari. A unified linear-time framework for sentence-level discourse parsing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4190–4200, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1410.
- [50] Kai Sheng Tai, Richard Socher, and Christopher D. Manning. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1556–1566, Beijing, China, July 2015. Association for Computational Linguistics. doi: 10.3115/v1/P15-1150. URL <https://www.aclweb.org/anthology/P15-1150>.
- [51] Akiko Eriguchi, Kazuma Hashimoto, and Yoshimasa Tsuruoka. Tree-to-sequence attentional neural machine translation. In *Proceedings of the 54th Annual Meeting*

- of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 823–833, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1078. URL <https://www.aclweb.org/anthology/P16-1078>.
- [52] Xuan-Phi Nguyen, Shafiq Joty, Steven Hoi, and Richard Socher. Tree-structured attention with hierarchical accumulation. In *International Conference on Learning Representations*, 2020. URL <https://openreview.net/forum?id=HJxK5pEYvr>.
- [53] Mitchell Stern, Daniel Fried, and Dan Klein. Effective inference for generative neural parsing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1695–1700, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/D17-1178. URL <https://www.aclweb.org/anthology/D17-1178>.
- [54] Michael Collins. Three generative, lexicalised models for statistical parsing. In *35th Annual Meeting of the Association for Computational Linguistics and 8th Conference of the European Chapter of the Association for Computational Linguistics*, pages 16–23, Madrid, Spain, July 1997. Association for Computational Linguistics. doi: 10.3115/976909.979620. URL <https://www.aclweb.org/anthology/P97-1003>.
- [55] Eugene Charniak. A maximum-entropy-inspired parser. In *1st Meeting of the North American Chapter of the Association for Computational Linguistics*, 2000. URL <https://www.aclweb.org/anthology/A00-2018>.
- [56] Dan Klein and Christopher D. Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 423–430, Sapporo, Japan, July 2003. Association for Computational Linguistics. doi: 10.3115/1075096.1075150. URL <https://www.aclweb.org/anthology/P03-1054>.
- [57] Slav Petrov and Dan Klein. Improved inference for unlexicalized parsing. In *Human Language Technologies 2007: The Conference of the North American Chapter of*

- the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 404–411, Rochester, New York, April 2007. Association for Computational Linguistics. URL <https://www.aclweb.org/anthology/N07-1051>.
- [58] David Hall, Greg Durrett, and Dan Klein. Less grammar, more features. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 228–237, Baltimore, Maryland, June 2014. Association for Computational Linguistics. doi: 10.3115/v1/P14-1022. URL <https://www.aclweb.org/anthology/P14-1022>.
- [59] Mitchell Stern, Jacob Andreas, and Dan Klein. A minimal span-based neural constituency parser. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 818–827, 2017.
- [60] Yu Zhang, Houquan Zhou, and Zhenghua Li. Fast and accurate neural crf constituency parsing. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 4046–4053. International Joint Conferences on Artificial Intelligence Organization, 7 2020. doi: 10.24963/ijcai.2020/560. Main track.
- [61] Maximin Coavoux and Benoît Crabbé. Neural greedy constituent parsing with dynamic oracles. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 172–182, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1017. URL <https://www.aclweb.org/anthology/P16-1017>.
- [62] Carlos Gómez, Rodríguez and David Vilares. Constituent parsing as sequence labeling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1314–1324. Association for Computational Linguistics, October–November 2018. doi: 10.18653/v1/D18-1162. URL <https://www.aclweb.org/anthology/D18-1162>.
- [63] Yikang Shen, Zhouhan Lin, Athul Paul Jacob, Alessandro Sordoni, Aaron Courville, and Yoshua Bengio. Straight to the tree: Constituency parsing with neural syntactic

- distance. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1171–1180, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1108.
- [64] Oriol Vinyals, Łukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. Grammar as a foreign language. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28, pages 2773–2781. Curran Associates, Inc., 2015. URL <https://proceedings.neurips.cc/paper/2015/file/277281aada22045c03945dcb2ca6f2ec-Paper.pdf>.
- [65] Hidetaka Kamigaito, Katsuhiko Hayashi, Tsutomu Hirao, Hiroya Takamura, Manabu Okumura, and Masaaki Nagata. Supervised attention for sequence-to-sequence constituency parsing. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 7–12, Taipei, Taiwan, November 2017. Asian Federation of Natural Language Processing. URL <https://www.aclweb.org/anthology/I17-2002>.
- [66] Jun Suzuki, Sho Takase, Hidetaka Kamigaito, Makoto Morishita, and Masaaki Nagata. An empirical study of building a strong baseline for constituency parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 612–618, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-2097. URL <https://www.aclweb.org/anthology/P18-2097>.
- [67] Daniel Fernández-González and Carlos Gómez-Rodríguez. Enriched in-order linearization for faster sequence-to-sequence constituent parsing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4092–4099, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.376. URL <https://www.aclweb.org/anthology/2020.acl-main.376>.
- [68] Junru Zhou and Hai Zhao. Head-driven phrase structure grammar parsing on penn treebank. In *Proceedings of the 57th Conference of the Association for Computational*

- Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 2396–2408, 2019. URL <https://www.aclweb.org/anthology/P19-1230/>.
- [69] Radu Soricut and Daniel Marcu. Sentence level discourse parsing using syntactic and lexical information. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 228–235, 2003.
- [70] Shafiq R. Joty and Alessandro Moschitti. Discriminative reranking of discourse parses using tree kernels. In *EMNLP*, pages 2049–2060. ACL, 2014.
- [71] Daniel Marcu. A Decision-based Approach to Rhetorical Parsing. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, ACL’99, pages 365–372, Morristown, NJ, USA, 1999. ACL.
- [72] Kenji Sagae. Analysis of discourse structure with syntactic dependencies and data-driven shift-reduce parsing. In *IWPT*, pages 81–84. The Association for Computational Linguistics, 2009.
- [73] Yizhong Wang, Sujian Li, and Houfeng Wang. A two-stage parsing method for text-level discourse analysis. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 184–188. Association for Computational Linguistics, 2017.
- [74] Hugo Hernault, Helmut Prendinger, David A. duVerle, and Mitsuru Ishizuka. Hilda: A discourse parser using support vector machine classification. *D&D*, 1(3):1–33, 2010.
- [75] Shafiq Joty, Giuseppe Carenini, Raymond Ng, and Yashar Mehdad. Combining intra- and multi-sentential rhetorical parsing for document-level discourse analysis. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 486–496, Sofia, Bulgaria, August 2013. Association for Computational Linguistics. URL <https://www.aclweb.org/anthology/P13-1048>.

- [76] Vanessa Wei Feng and Graeme Hirst. A linear-time bottom-up discourse parser with constraints and post-editing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 511–521, Baltimore, Maryland, June 2014. Association for Computational Linguistics. doi: 10.3115/v1/P14-1048. URL <https://www.aclweb.org/anthology/P14-1048>.
- [77] Yang Liu and Mirella Lapata. Learning contextually informed representations for linear-time discourse parsing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1289–1298, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/D17-1133. URL <https://www.aclweb.org/anthology/D17-1133>.
- [78] Daniel Marcu. The rhetorical parsing of unrestricted texts: a surface-based approach. *Computational Linguistics*, 26(3):395–448, 2000. URL <https://www.aclweb.org/anthology/J00-3005>.
- [79] Thanh-Tung Nguyen, Xuan-Phi Nguyen, Shafiq Joty, and Xiaoli Li. Efficient constituency parsing by pointing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3284–3294, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.301. URL <https://www.aclweb.org/anthology/2020.acl-main.301>.
- [80] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL <https://www.aclweb.org/anthology/N19-1423>.

- [81] Jenny Rose Finkel, Alex Kleeman, and Christopher D. Manning. Efficient, feature-based, conditional random field parsing. In *Proceedings of ACL-08: HLT*, pages 959–967, Columbus, Ohio, June 2008. Association for Computational Linguistics. URL <https://www.aclweb.org/anthology/P08-1109>.
- [82] Eugene Charniak and Mark Johnson. Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL’05)*, pages 173–180, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics. doi: 10.3115/1219840.1219862. URL <https://www.aclweb.org/anthology/P05-1022>.
- [83] Kenji Sagae and Alon Lavie. A classifier-based parser with linear run-time complexity. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 125–132, Vancouver, British Columbia, October 2005. Association for Computational Linguistics. URL <https://www.aclweb.org/anthology/W05-1513>.
- [84] Nikita Kitaev and Dan Klein. Tetra-tagging: Word-synchronous parsing with linear-time inference. *CoRR*, abs/1904.09745, 2019. URL <http://arxiv.org/abs/1904.09745>.
- [85] Daniel Fernández-González and André F. T. Martins. Parsing as reduction. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1523–1533, Beijing, China, July 2015. Association for Computational Linguistics. doi: 10.3115/v1/P15-1147. URL <https://www.aclweb.org/anthology/P15-1147>.
- [86] Oriol Vinyals, Łukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. Grammar as a foreign language. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2773–2781. Curran Associates, Inc., 2015. URL <http://papers.nips.cc/paper/5635-grammar-as-a-foreign-language.pdf>.

- [87] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017. URL <http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf>.
- [88] Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Comput. Linguist.*, 19(2):313–330, June 1993. ISSN 0891-2017. URL <http://dl.acm.org/citation.cfm?id=972470.972475>.
- [89] Djamé Seddah, Reut Tsarfaty, Sandra Kübler, Marie Candito, Jinho D. Choi, Richárd Farkas, Jennifer Foster, Iakes Goenaga, Koldo Gojenola Gallettebeitia, Yoav Goldberg, Spence Green, Nizar Habash, Marco Kuhlmann, Wolfgang Maier, Joakim Nivre, Adam Przepiórkowski, Ryan Roth, Wolfgang Seeker, Yannick Versley, Veronika Vincze, Marcin Woliński, Alina Wróblewska, and Eric Villemonte de la Clergerie. Overview of the SPMRL 2013 shared task: A cross-framework evaluation of parsing morphologically rich languages. In *Proceedings of the Fourth Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 146–182, Seattle, Washington, USA, October 2013. Association for Computational Linguistics. URL <https://www.aclweb.org/anthology/W13-4917>.
- [90] Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1, NAACL '03*, pages 173–180. Association for Computational Linguistics, 2003. doi: 10.3115/1073445.1073478. URL <https://doi.org/10.3115/1073445.1073478>.
- [91] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

- [92] Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. Fast and accurate shift-reduce constituent parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, pages 434–443. Association for Computational Linguistics, August 2013. URL <https://www.aclweb.org/anthology/P13-1043>.
- [93] Anders Bjorkelund, Ozlem Cetinoglu, Agnieszka Falenska, Richard Farkas, Thomas Mueller, Wolfgang Seeker, and Zsolt Szanto. The ims-wroclaw-szeged-cis entry at the spmrl 2014 shared task: Reranking and morphosyntax meet unlabeled data. In *Proceedings of the First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of NonCanonical Languages*, pages 97–102, 2014.
- [94] Maximin Coavoux and Benoît Crabbé. Multilingual lexicalized constituency parsing with word-level auxiliary tasks. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 331–336, Valencia, Spain, April 2017. Association for Computational Linguistics. URL <https://www.aclweb.org/anthology/E17-2053>.
- [95] Thanh-Tung Nguyen, Xuan-Phi Nguyen, Shafiq Joty, and Xiaoli Li. A conditional splitting framework for efficient constituency parsing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages xx–xx, Online, August 2021. Association for Computational Linguistics.
- [96] Daniel Fernández-González and Carlos Gómez-Rodríguez. Left-to-right dependency parsing with pointer networks. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 710–716, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1076. URL <https://www.aclweb.org/anthology/N19-1076>.
- [97] Timothy Dozat and Christopher D. Manning. Deep biaffine attention for neural dependency parsing. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

- [98] Yang Wei, Yuanbin Wu, and Man Lan. A span-based linearization for constituent trees. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3267–3277, Online, July 2020. Association for Computational Linguistics.
- [99] Thanh-Tung Nguyen, Xuan-Phi Nguyen, Shafiq Joty, and Xiaoli Li. Rst parsing from scratch. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages xx–xx, Online, June 2021. Association for Computational Linguistics.
- [100] Yangfeng Ji and Noah A. Smith. Neural discourse structure for text categorization. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 996–1005, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1092. URL <https://www.aclweb.org/anthology/P17-1092>.
- [101] Shima Gerani, Yashar Mehdad, Giuseppe Carenini, Raymond T. Ng, and Bitia Nejat. Abstractive summarization of product reviews using discourse structure. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1602–1613, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1168. URL <https://www.aclweb.org/anthology/D14-1168>.
- [102] Parminder Bhatia, Yangfeng Ji, and Jacob Eisenstein. Better document-level sentiment analysis from RST discourse parsing. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2212–2218, Lisbon, Portugal, September 2015. Association for Computational Linguistics. doi: 10.18653/v1/D15-1263. URL <https://www.aclweb.org/anthology/D15-1263>.
- [103] Shafiq Joty, Francisco Guzmán, Lluís Màrquez, and Preslav Nakov. Discourse structure in machine translation evaluation. *Computational Linguistics*, 43(4):683–722, December 2017. doi: 10.1162/COLI_a_00298. URL <https://www.aclweb.org/anthology/J17-4001>.

- [104] Yifan Gao, Chien-Sheng Wu, Jingjing Li, Shafiq Joty, Steven C.H. Hoi, Caiming Xiong, Irwin King, and Michael Lyu. Discern: Discourse-aware entailment reasoning network for conversational machine reading. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2439–2449, Online, November 2020. Association for Computational Linguistics. URL <https://www.aclweb.org/anthology/2020.emnlp-main.191>.
- [105] William C Mann and Sandra A Thompson. Rhetorical structure theory: Toward a functional theory of text organization. *Text*, 8(3):243–281, 1988.
- [106] Chloé Braud, Maximin Coavoux, and Anders Søgaard. Cross-lingual RST discourse parsing. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 292–304, Valencia, Spain, April 2017. Association for Computational Linguistics. URL <https://www.aclweb.org/anthology/E17-1028>.
- [107] Yizhong Wang, Sujian Li, and Houfeng Wang. A two-stage parsing method for text-level discourse analysis. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 184–188, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-2029. URL <https://www.aclweb.org/anthology/P17-2029>.
- [108] Hugo Hernault, Helmut Prendinger, David A. duVerle, and Mitsuru Ishizuka. Hilda: A discourse parser using support vector machine classification. *Dialogue and Discourse*, 1(3):1–33, 2010.
- [109] Chloé Braud, Barbara Plank, and Anders Søgaard. Multi-view and multi-task training of RST discourse parsers. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 1903–1913, Osaka, Japan, December 2016. The COLING 2016 Organizing Committee. URL <https://www.aclweb.org/anthology/C16-1179>.

- [110] Shafiq Joty, Giuseppe Carenini, and Raymond Ng. A novel discriminative framework for sentence-level discourse analysis. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 904–915, Jeju Island, Korea, July 2012. Association for Computational Linguistics.
- [111] Zhengyuan Liu, Ke Shi, and Nancy Chen. Multilingual neural RST discourse parsing. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6730–6738, Barcelona, Spain (Online), December 2020. International Committee on Computational Linguistics. doi: 10.18653/v1/2020.coling-main.591. URL <https://www.aclweb.org/anthology/2020.coling-main.591>.
- [112] Kai Zhao and Liang Huang. Joint syntacto-discourse parsing and the syntacto-discourse treebank. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2117–2123, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/D17-1225. URL <https://www.aclweb.org/anthology/D17-1225>.
- [113] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL <http://www.aclweb.org/anthology/D14-1162>.
- [114] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.
- [115] Seeger Fisher and Brian Roark. The utility of parse-derived features for automatic discourse segmentation. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 488–495, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL <https://www.aclweb.org/anthology/P07-1062>.

- [116] Carlson Lynn, Daniel Marcu, and Mary Ellen Okurowski. Rst discourse treebank (rst-dt) ldc2002t07. *Linguistic Data Consortium*, 2002.
- [117] Mathieu Morey, Philippe Muller, and Nicholas Asher. A dependency perspective on RST discourse parsing and evaluation. *Computational Linguistics*, 44(2):197–235, June 2018. doi: 10.1162/COLI_a_00314. URL <https://www.aclweb.org/anthology/J18-2001>.
- [118] Lynn Carlson and Daniel Marcu. Discourse tagging reference manual. Technical Report ISI-TR-545, University of Southern California Information Sciences Institute, 2001. URL <http://www.isi.edu/marcu/discourse/tagging-ref-manual.pdf>.
- [119] Qi Li, Tianshi Li, and Baobao Chang. Discourse parsing with attention-based hierarchical neural networks. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 362–371, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1035. URL <https://www.aclweb.org/anthology/D16-1035>.
- [120] Jing Li, Aixin Sun, and Shafiq Joty. Segbot: A generic neural text segmentation model with pointer network. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 4166–4172. International Joint Conferences on Artificial Intelligence Organization, 7 2018. doi: 10.24963/ijcai.2018/579. URL <https://doi.org/10.24963/ijcai.2018/579>.
- [121] Amir Zeldes, Debopam Das, Erick Galani Maziero, Juliano Antonio, and Mikel Iruskieta. The DISRPT 2019 shared task on elementary discourse unit segmentation and connective detection. In *Proceedings of the Workshop on Discourse Relation Parsing and Treebanking 2019*, pages 97–104, Minneapolis, MN, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-2713. URL <https://www.aclweb.org/anthology/W19-2713>.
- [122] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv:2004.05150*, 2020.

- [123] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019. URL <https://openreview.net/forum?id=HyzdRiR9Y7>.
- [124] Thanh-Tung Nguyen, Xuan-Phi Nguyen, Shafiq Joty, and Xiaoli Li. Differentiable window for dynamic local attention. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6589–6599, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.589. URL <https://www.aclweb.org/anthology/2020.acl-main.589>.
- [125] Kelvin Xu, Jimmy Lei Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML’15, pages 2048–2057. JMLR.org, 2015. URL <http://dl.acm.org/citation.cfm?id=3045118.3045336>.
- [126] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2048–2057, Lille, France, 07–09 Jul 2015. PMLR. URL <http://proceedings.mlr.press/v37/xuc15.html>.
- [127] Matthias Sperber, Jan Niehues, Graham Neubig, Sebastian Stuker, and Alex Waibel. Self-attentional acoustic models. In *Interspeech 2018, 19th Annual Conference of the International Speech Communication Association, Hyderabad, India, 2-6 September 2018.*, pages 3723–3727, 2018. doi: 10.21437/Interspeech.2018-1910. URL <https://doi.org/10.21437/Interspeech.2018-1910>.
- [128] Baosong Yang, Zhaopeng Tu, Derek F. Wong, Fandong Meng, Lidia S. Chao, and Tong Zhang. Modeling localness for self-attention networks. In *Proceedings of the*

- 2018 *Conference on Empirical Methods in Natural Language Processing*, pages 4449–4458, Brussels, Belgium, October–November 2018. Association for Computational Linguistics. URL <https://www.aclweb.org/anthology/D18-1475>.
- [129] Yonatan Belinkov, Nadir Durrani, Fahim Dalvi, Hassan Sajjad, and James Glass. What do neural machine translation models learn about morphology? In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 861–872, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1080. URL <https://www.aclweb.org/anthology/P17-1080>.
- [130] Matthew Peters, Mark Neumann, Luke Zettlemoyer, and Wen-tau Yih. Dissecting contextual word embeddings: Architecture and representation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1499–1509, Brussels, Belgium, October–November 2018. Association for Computational Linguistics. URL <https://www.aclweb.org/anthology/D18-1179>.
- [131] Ganesh Jawahar, Benoît Sagot, and Djamé Seddah. What does BERT learn about the structure of language? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3651–3657, Florence, Italy, July 2019. Association for Computational Linguistics. URL <https://www.aclweb.org/anthology/P19-1356>.
- [132] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725. Association for Computational Linguistics, 2016. doi: 10.18653/v1/P16-1162. URL <http://www.aclweb.org/anthology/P16-1162>.
- [133] Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. Scaling neural machine translation. In *Proceedings of the Third Conference on Machine Translation (WMT)*, 2018.

- [134] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642. Association for Computational Linguistics, 2013. URL <http://aclweb.org/anthology/D13-1170>.
- [135] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL <http://www.aclweb.org/anthology/P11-1015>.
- [136] Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. Assessing the ability of lstms to learn syntax-sensitive dependencies. *Transactions of the Association for Computational Linguistics*, 4:521–535, 2016. ISSN 2307-387X. URL <https://transacl.org/ojs/index.php/tacl/article/view/972>.
- [137] Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. One billion word benchmark for measuring progress in statistical language modeling. Technical report, Google, 2013. URL <http://arxiv.org/abs/1312.3005>.
- [138] Steven Cao, Nikita Kitaev, and Dan Klein. Unsupervised parsing via constituency tests. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4798–4808, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.389. URL <https://www.aclweb.org/anthology/2020.emnlp-main.389>.