Computational Models of Word Order

Von der Fakultät Informatik, Elektrotechnik und Informationstechnik der Universität Stuttgart zur Erlangung der Würde eines Doktors der Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung.

Vorgelegt von

Xiang Yu

aus Hunan, China

Hauptberichter:	Prof. Dr. Jonas Kuhn
1. Mitberichter:	Prof. Dr. Ngoc Thang Vu
2. Mitberichter:	Prof. Dr. Leo Wanner

Tag der mündlichen Prüfung:	07.06.2021

Institut für Maschinelle Sprachverarbeitung der Universität Stuttgart

2022

I hereby declare that I have created this work completely on my own and used no other sources or tools than the ones listed, and that I have marked any citations accordingly.

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Berlin, May 20, 2022

Place, Date

Xiang Yu

Contents

Abstract xvii

Überblick xix

1 Introduction 1

1.1 Research Gaps and Contribution . . . 1
1.2 Outline . . . 3
1.3 Publications . . . 5

2 Background 9

2.1 Dependency Grammar . . . 9
2.1.1 Dependency Grammar vs. Constituency Grammar . . . 9
2.1.2 Projectivity . . . 11
2.1.3 Parsing and Linearization . . . 12
2.2 Dependency Parsing Methods . . . 13
2.2.1 Graph-based Parsing . . . 13
2.2.2 Transition-based Parsing . . . 14
2.2.3 Non-Projective Parsing . . . 14
2.3 Linearization Methods . . . 15
2.3.1 Syntax-free Linearization . . . 16
2.3.2 Syntax-based Linearization . . . 16
2.4 Machine Learning Models . . . 18
2.4.1 Perceptron . . . 18
2.4.2 Multi-Layer Perceptron . . . 18
2.4.3 Distributed Representation . . . 19
2.4.4 Recurrent Neural Networks . . . 20
2.4.5 Sequence-to-Sequence Model and Attention Mechanism . . . 21

2.4.6 Tree-LSTM . . . 23
2.4.7 Graph Neural Networks . . . 24
2.5 Dependency Length Minimization . . . 25

3 Dependency Tree Linearization **27**
3.1 Encoding . . . 29
3.1.1 Token . . . 29
3.1.2 Sequence . . . 30
3.1.3 Tree-LSTM . . . 31
3.1.4 Graph Neural Networks . . . 33
3.2 Divide-and-Conquer Strategy . . . 34
3.3 Decoding . . . 35
3.3.1 Incremental Decoding . . . 35
Left-to-Right Generation with Beam Search . . . 35
Head-First Generation Order . . . 36
Training with Beam Search . . . 37
Ensemble . . . 39
3.3.2 Graph-based Decoding . . . 39
Scoring Model . . . 40
TSP Solver . . . 41
Training Objective . . . 42
Decoding with Constraints . . . 43
3.3.3 Transition-based Decoding . . . 44
Transition System . . . 44
Scoring Model . . . 44
Producing Non-Projective Sentences . . . 45
3.4 Summary . . . 48

4 Linearization Experiments **51**
4.1 Data and Evaluation Metrics . . . 51
4.2 Encoders . . . 54
4.3 Decoders . . . 56
4.4 Model Errors vs. Search Errors . . . 58
4.5 Non-Projectiviy . . . 59
4.6 Training Data Sizes . . . 61
4.7 Error Analysis . . . 62

4.8 Case Studies . . . 68
4.8.1 English . . . 69
4.8.2 German . . . 70
4.8.3 Chinese . . . 71
4.9 Summary . . . 72

5 Dependency Length Minimization 75
5.1 Experimental Settings . . . 76
5.1.1 Data . . . 76
5.1.2 Baselines . . . 76
5.1.3 Baseline Dependency Lengths . . . 77
5.2 Analyses . . . 78
5.2.1 Word Order Constraints . . . 78
5.2.2 Dependent Distribution: FREE vs. VAL . . . 80
5.2.3 Head Direction: VAL vs. SIDE . . . 81
5.2.4 Sibling Ordering: SIDE vs. OBS . . . 83
5.3 Experiments with Linearizer . . . 84
5.4 DLM and Non-Projectivity . . . 85
5.5 Summary . . . 88

6 Surface Realization Shared Tasks 89
6.1 Task Description . . . 89
6.2 Full Pipeline . . . 91
6.2.1 Linearization . . . 93
6.2.2 Function Word Completion . . . 93
6.2.3 Morphological Inflection . . . 95
6.2.4 Contraction . . . 96
6.2.5 Detokenization . . . 96
6.3 Data Augmentation . . . 97
6.4 Ensemble . . . 98
6.5 Shared Task Results . . . 99
6.5.1 Evaluation Metrics . . . 99
6.5.2 SR’19 Results . . . 99
6.5.3 SR’20 Results . . . 101
6.6 Summary . . . 104

7 Conclusion **105**
7.1 Contributions . . . 105
7.2 Future Perspectives . . . 107

A Detailed Experimental Results **109**

Bibliography **119**

List of Figures

2.1 Examples of constituency tree and dependency tree. 10

2.2 Dependency trees of the same meaning in different languages. 11

2.3 An example non-projective sentence. 11

3.1 A full pipeline for the surface realization shared task, including five modules: (1) linearization, (2) completion, (3) inflection, (4) contraction, and (5) detokenization. 28

3.2 Different ways to serialize a dependency tree as a sequence. 31

3.3 A schematic illustration of the bidirectional Tree-LSTM. 32

3.4 An illustration of the information flow in the encoder, where the red dotted arrows represent the bottom-up pass and the blue dashed arrows represent the top-down pass. The solid arrows illustrate the information flow from node 8 to node 4. 33

3.5 Multiple possible full tree orders from the same subtree orders, where (a) and (b) are ordered subtrees, (c), (d), and (e) are possible full ordered trees, among which only (c) is projective. 34

3.6 The matrix and graph views of the linearization task as a TSP. 40

3.7 An illustration of the swap model. (1) shows a configuration [1 2 | 3 4 5 6], where the stack contains 1 and 2, the buffer contains 3, 4, 5, and 6, and the model predicts *shift* to move 3 from the buffer to the stack; (2) is the resulted configuration, and the model predicts *swap*, which moves 3 back to the buffer behind 4, as shown in (3). The solid black arrows represent the computation already done in the previous steps, and the dashed red arrows represent the new computation needed for the current step. 45

3.8 An pair of projective and non-projective sentences. 46

3.9 An overview of the optimal linearization system. 49

4.1 An example to demonstrate the different evaluation metrics, where (a) is the reference sentence and (b) is the predicted sentence. 54
4.2 The improvement of full tree match after reordering with respect to the percentage of non-projective arcs in each treebank. 60
4.3 BLEU score with respect to the sentence length and the non-projective arc rate. 61
4.4 BLEU scores of models trained on different data sizes (on a logarithmic scale), where (a) shows the average and (b) shows individual treebanks. . 62
4.5 An example of two types of error cases, where the correct order should be [1, 2, 3, 4, 5, 6], head errors are marked with red (d and e), sibling errors are marked with blue (b and c). 63
4.6 Frequency of each dependency relation in each language, where the treebanks are grouped by the language families and subfamilies, and the relations are grouped by the structural categories of the dependent. 64
4.7 Head and sibling error rates of each dependency relation in each language. 65
4.8 Difference of errors between the lexicalized model and the delexicalized model, which illustrates the benefit of lexicalization. 67

5.1 Example dependency trees and their dependency lengths, adapted from Futrell et al. (2015a). The second sentence is preferred since it has shorter DL and lower cognitive burden. 76
5.2 An example tree and results of four baseline linearization methods, where the label represents the size of the subtree. 77
5.3 Average DL vs. sentence length across 53 treebanks. 78
5.4 All treebanks characterized by the two types of word order freedom. . . . 79
5.5 Percentage of trees where the real order has shorter DL than flipping one dependent. 82
5.6 Percentage of trees where the real order has shorter DL than swapping two siblings. 83
5.7 Average DL vs. sentence length across 53 treebanks. The additional results of DELEX and STRUCT are from the trained linearization models, while the others are directly from manipulation of the original treebank. 84
5.8 The percentage of non-projective trees and the DL of the fully random linearization, compared to the projective random linearization and the real data. 86

5.9 A pair of projective and non-projective sentence in German, and their DLs. The non-projective sentence with the extraposed relative clause has shorter DL. 87
5.10 The comparisons between the non-projective sentences and their projectivized versions in terms of the DL. 87

6.1 A full pipeline for the surface realization shared task, including five modules: (1) linearization, (2) completion, (3) inflection, (4) contraction, and (5) detokenization. 92
6.2 An example of the completion process to the right side, where the right arrows illustrate the forward LSTM, and the left arrows the backward LSTM. 94
6.3 An illustration of the contraction model. 97
6.4 Sentence length distribution in en_lines before and after filtering. 98

List of Tables

2.1 Transitions of the ARCSTANDARD system. 14

2.2 Transitions of the ARCSTANDARDSWAP system. 15

3.1 An example sentence in the CoNLL-U format. 29

3.2 The SWAP transition system. 44

3.3 The transitions to reorder the projective tree in Figure 3.8. 47

4.1 Information about the treebanks used in the experiments, including the language (sub-)family, data size, average sentence length, average non-leaf subtree size, percentage of non-projective sentences and non-projective arcs. 52

4.2 Different encoders combined the subtree and full tree TSP decoders, evaluated on the BLEU score and the exact matching rates of subtrees and full trees. 55

4.3 Different versions of the sequence encoders combined the subtree and full tree TSP decoders, evaluated on the BLEU score and the exact matching rates of subtrees and full trees. 56

4.4 Performances of different decoders on the subtree and full tree level. . . . 57

4.5 Model and search error rates by the `TSP-sub` and `TSP-full` decoders. . 58

4.6 TSP decoders on the full tree and subtree level and additionally with reordering, evaluated on the BLEU score and the exact matching rates of subtrees and full trees. 59

4.7 Percentage of sentences that are changed better or worse by the reordering. 60

4.8 Distribution of deviation categories in the prediction. 69

6.1 Numbers of sentences of the datasets in SR'19 and SR'20, where the in-domain, out-of-domain, and automatically parsed treebanks are used in both shared tasks, and the automatically parsed Wikipedia treebanks are newly introduced in SR'20. 90
6.2 An example sentence for the shallow track (T1). 91
6.3 An example sentence for the deep track (T2). 91
6.4 Automatic Evaluation Results of the shallow track (T1) and the BLEU difference with the best system among other participants for each treebank. . 100
6.5 Automatic Evaluation Results of the deep track (T2) and the BLEU difference with the best system among other participants for each treebank. . . 101
6.6 Comparing the average BLEU scores of our SR'19 results and our SR'20 results in the closed (2020a) and open (2020b) subtrack. 101
6.7 Automatic evaluation results in the shallow track (T1) of the our system in SR'19 and the closed subtrack (2020a) and open subtrack (2020b) in SR'20. 102
6.8 Automatic evaluation results in the deep track (T2) of the our system in SR'19 and the closed subtrack (2020a) and open subtrack (2020b) in SR'20. 103

A.1 Detailed BLEU scores for the experiment on encoders in §4.2. 110
A.2 Detailed EMF scores for the experiment on encoders in §4.2. 111
A.3 Detailed EMS scores for the experiment on encoders in §4.2. 112
A.4 Detailed BLEU scores for the experiment on decoders in §4.3. 113
A.5 Detailed EMF scores for the experiment on decoders in §4.3. 114
A.6 Detailed EMS scores for the experiment on decoders in §4.3. 115
A.7 Detailed BLEU scores for the experiment on training data sizes in §4.6. . . 116
A.8 Detailed EMF scores for the experiment on training data sizes in §4.6. . . . 117
A.9 Detailed EMS scores for the experiment on training data sizes in §4.6. . . . 118

List of Abbreviations

AMR	Abstract Meaning Representation
DL	Dependency Length
DLM	Dependency Length Minimization
GAT	Graph Attention Network
GCN	Graph Convolutional Network
GNN	Graph Neural Network
LSTM	Long Short-Term Memory
MLP	Multi-layer Perceptron
MST	Maximum Spanning Tree
NLP	Natural Language Processing
RNN	Recurrent Neural Network
Seq2Seq	Sequence-to-Sequence
TSP	Traveling Salesman Problem
UD	Universal Dependencies
UPOS	Universal Part-of-Speech
VRP	Vehicle Routing Problem
XPOS	Language-specific Part-of-Speech

Abstract

A sentence in our mind is not a simple sequence of words but a hierarchical structure. We put the sentence in the linear order when we utter it for communication. **Linearization** is the task of mapping the hierarchical structure of a sentence into its linear order.

Our work is based on the **dependency grammar**, which models the dependency relation between the words, and the resulting syntactic representation is a directed tree structure. The popularity of dependency grammar in Natural Language Processing (NLP) benefits from its separation of structure order and linear order and its emphasis on syntactic functions. These properties facilitate a universal annotation scheme covering a wide range of languages used in our experiments.

We focus on developing a **robust** and **efficient** computational model that finds the linear order of a dependency tree. We take advantage of deep learning models' expressive power to encode the syntactic structures of typologically diverse languages robustly. We take a graph-based approach that combines a simple bigram scoring model and a greedy decoding algorithm to search for the optimal word order efficiently. We use the divide-and-conquer strategy to reduce the search space, which restricts the output to be projective. We then resolve the restriction with a transition-based post-processing model.

Apart from the computational models, we also study the word order from a quantitative linguistic perspective. We examine the Dependency Length Minimization (DLM) hypothesis, which is believed to be a universal factor that affects the word order of every language. It states that human languages tend to order the words to minimize the overall length of dependency arcs, which reduces the cognitive burden of speaking and understanding. We demonstrate that DLM can explain every aspect of word order in a dependency tree, such as the direction of the head, the arrangement of sibling dependents, and the existence of crossing arcs (non-projectivity). Furthermore, we find that DLM not only shapes the general word order preferences but also motivates the occasional deviation from the preferences.

Finally, we apply our model in the task of surface realization, which aims to generate a sentence from a deep syntactic representation. We implement a pipeline with five steps, (1) linearization, (2) function word generation, (3) morphological inflection, (4) contraction, and (5) detokenization, which achieved state-of-the-art performance.

Überblick

Ein Satz ist in unserem Kopf keine einfache Folge von Wörtern, sondern eine hierarchische Struktur. Wir bringen den Satz in die lineare Reihenfolge, wenn wir ihn zur Kommunikation aussprechen. **Linearisierung** ist die Aufgabe, die hierarchische Struktur eines Satzes in seine lineare Ordnung abzubilden.

Unsere Arbeit basiert auf der **Dependenzgrammatik**, die die Dependenz (Abhängigkeitsbeziehung) zwischen den Wörtern modelliert, und die resultierende syntaktische Darstellung ist eine gerichtete Baumstruktur. Die Popularität der Dependenzgrammatik in der Machinellen Sprachverarbeitung profitiert von ihrer Trennung von struktureller und linearer Ordnung und ihrer Betonung auf der syntaktischen Funktionen. Diese Eigenschaften ermöglichen ein universelles Annotationsschema, das eine Vielzahl von Sprachen abdeckt, die in unseren Experimenten verwendet werden.

In dieser Arbeit konzentrieren wir uns auf die Entwicklung eines **robusten** und **effizienten** Berechnungsmodells, das die lineare Ordnung eines Dependenzbaums findet. Wir nutzen die Ausdruckskraft von Deep-Learning-Modellen, um die syntaktischen Strukturen von typologisch unterschiedlichen Sprachen robust zu kodieren. Wir verwenden einen graphenbasierten Ansatz, der ein einfaches Bigram-Scoring-Modell und einen Greedy-Decodierungsalgorithmus kombiniert, um effizient nach der optimalen Wortreihenfolge zu suchen. Wir verwenden die Divide-and-Conquer-Strategie, um den Suchraum zu reduzieren, was die Ausgabe auf projektive Dependenzbäume beschränkt. Anschließend lösen wir die Einschränkung mit einem transitionsbasierten Nachbearbeitungsmodell auf.

Zusätzlich zu den Berechnungsmodellen untersuchen wir die Wortreihenfolge auch aus einer quantitativen linguistischen Perspektive. Wir untersuchen die Dependency-Length-Minimization (DLM)-Hypothese, von der angenommen wird, dass sie ein universeller Faktor ist, der die Wortreihenfolge jeder Sprache beeinflusst. Sie besagt, dass menschliche Sprachen dazu neigen, die Wörter so anzuordnen, dass die Gesamtlänge der Dependenzkanten minimiert wird, was die kognitive Belastung beim Sprechen und Verstehen reduziert. Wir zeigen, dass DLM jeden Aspekt der Wortreihenfolge in einem Dependenzbaum erklären kann, wie z. B. die Richtung des Kopfes, die Anordnung von Geschwister-Dependenten und die Existenz von kreuzenden Dependenzkanten (Nicht-Projektivität). Außerdem finden wir, dass DLM nicht nur die allgemeinen Wortordnungspräferenzen formt, sondern auch die gelegentliche Abweichung von den Präferenzen motiviert.

Schließlich wenden wir unser Modell bei der Aufgabe der Oberflächenrealisierung an, die darauf abzielt, einen Satz aus einer tiefen syntaktischen Repräsentation zu erzeugen. Wir implementieren eine Pipeline mit fünf Schritten, (1) Linearisierung, (2) Funktionsworterzeugung, (3) morphologische Flexion, (4) Kontraktion und (5) Detokenisierung, die eine State-of-the-Art-Leistung erzielt.

Acknowledgements

My journey in computational linguistics started with randomly picking up a novel *Story of your Life* by Ted Chiang in a library, which described an alien creature who has no linear conception of time and no linear word order in their language. At that point, I had no idea that I would have spent so many years studying language and ending up with a dissertation on word order. In retrospect, it was just like the story in the novel.

Throughout my journey at the IMS, my deepest gratitude goes to my advisor Jonas Kuhn. He offered me a research assistant job since I was an undergraduate student. Later I wrote my Bachelor's, Master's, and Doctoral Theses under his supervision. He has given me great freedom to explore the topics that I found interesting, and helped me finding the connection of ideas on a higher perspective and steering in uncertainty. Eventually, some seemingly random pieces of research work intertwined nicely together and resulted in this dissertation.

My other advisor Ngoc Thang Vu has also given me a great amount of guidance, and in a complementary way to Jonas. He focused on the technical side of machine learning, and provided hands-on support. We had many discussions on the details of my models, and spent time together debugging my code. He has always encouraged me with his great optimism when I was in doubt about my ideas.

Apart from my two advisors, I have also benefited tremendously from Wolfgang Seeker and Anders Björkelund, both co-supervisors of my student work and thesis, as well as my office mates. From them, I learned not only a great amount of knowledge about parsing and machine learning but also critical thinking in research. I am also very grateful to Agnieszka Faleńska for the cooperation in teaching and research, for plotting all the beautiful figures in our papers, and for being a cheerful and candid friend.

Also many thanks to all the friends and colleagues at IMS, with whom I have had countless discussions about research and life, Özlem Çetinoğlu, Kyle Richardson, Simon Tannert, Sean Papay, Daniel Ortega *et. al*. With all your company, I have enjoyed every day in my journey (except for those few sleepless days before conference deadlines).

A great portion of my dissertation is motivated by the work of two former colleagues from Stuttgart, Bernd Bohnet and Leo Wanner. Special thanks to Leo for reviewing my thesis as a committee member.

Finally, I would like to thank my parents for their constant support of my decisions, my wife Yang for putting up with me, and my son Luke, whose due date was a week after finishing the draft, for forcing me to eventually overcome my procrastination. And now begins the story of your life.

Chapter 1

Introduction

Dependency grammar is a grammar formalism based on the dependency relations between words, pioneered by Tesnière (1959, 2015). The dependency relations such as subject and object link the words in a sentence and build up a tree structure representing the meaning of the sentence. A special trait of dependency grammar is that it is agnostic to the surface order of the sentence. The linear order of the words in the sentence is not modeled by the dependency structure and is treated as a secondary role in the representation of the sentence, as stated in Tesnière (2015, p. 13), "[o]ne must not lose sight of the fact that syntactically, the true sentence is the structural sentence, for which the linear sentence is only an image projected onto the spoken chain, with all the disadvantages associated with flatness. "

In Tesnière's theory, when we utter a sentence, we first have a tree-structured representation in mind and transform that into the linear sequence of words by traversing the structure. This process is called **linearization**. The opposite process is called **parsing**; namely, we receive a sentence in its linear form and analyze its syntactic structure by identifying the dependency relations among the words.

We have two main focuses in this thesis. The first is to develop a computational model that determines the word order of a dependency tree; the second is to explore on a higher level the factor that universally influences the word order in all languages.

1.1 Research Gaps and Contribution

Developing a stable and efficient linearization model. Several linearization systems have been developed over the years, and some already achieved very strong performance, e.g., Bohnet et al. (2012). However, most of them are based on perceptron with feature templates, which work well on English but would struggle with morphologically rich or low-resource languages. Some linearizers are very slow due to the beam search decoding, so that they can not satisfy the speed requirements of real-time generation. In this thesis, we aim to develop a linearization system with stable multilingual performance thanks to the powerful representation ability of deep learning models while remaining highly efficient thanks to the simple decoding algorithm.

To achieve this goal, we will explore several linearization algorithms from different paradigms. The first algorithm generates the linear order incrementally by selecting the words from the input tree one after another. We use a Recurrent Neural Network (RNN), in particular, the Long Short-Term Memory (LSTM), to calculate the partial sequence score and use beam search to explore multiple hypotheses simultaneously. The second algorithm treats linearization as a Traveling Salesman Problem (TSP), namely to find a traversal path that visits each word exactly once. We use a biaffine model to calculate all possible bigrams' scores and use a greedy TSP solving algorithm to find the approximately optimal traversal path. The third algorithm is a transition system with a stack and a buffer similar to the transition-based parsing system. It uses two operations, *shift* to move the words from the buffer to the stack and *swap* to swap the order of two words. Unlike the previous two, where the input is a set of words, this algorithm takes a sequence as input and outputs another sequence.

Impact of dependency length minimization on word order. Apart from developing linearization models for each individual language, we are also interested in understanding what universally shapes the word order in all languages. It has been explored since Greenberg's Universal that the word order is more frequent to have a shorter distance between the head and the dependent (Greenberg, 1963). More recent work (Temperley, 2007; Liu, 2008; Futrell et al., 2015a) has focused on quantifying the dependency length and exploring how it affects word order, which is commonly referred to as the Dependency Length Minimization (DLM) hypothesis.

We follow this line of research to provide further evidence to the hypothesis. Furthermore, we break down the word order into more fine-grained factors, including the direction of the dependency and the pairwise arrangement of sibling dependents. With incremental baselines that start with the random word order, we gradually conform with the word order preferences toward the actual sentences in the treebank collections. In this way, we can explore the effect of DLM in each factor of word order. Moreover, we show that DLM not only shapes the general word order preferences in these factors, it also motivates the deviation from such preferences when necessary.

Treatment of non-projective sentences. **Projectivity** is a important property that emerges from the combination of linear order and structural order in a sentence. When the words in a sentence are linearly ordered and the dependency arcs do not cross each other, it is called a projective sentence or a projective tree; otherwise, it is non-projective. Most languages have very few non-projective sentences, such as English and Chinese, although theoretically, among all possible arrangements of words in a sentence, the vast majority are non-projective. Some languages with more flexible word order, such as Latin or Russian, tend to have a much higher percentage of non-projective sentences.

Many linearization algorithms and parsing algorithms assume that all the sentences they generate or parse are restricted to be projective. The main reason is that the projective assumption vastly reduces the search space; therefore, the projective algorithms typically have lower time complexity than the non-projective algorithms. Also, since most sentences are projective, it is more likely to find the correct parse tree or word order in the restricted search space. This is an engineering trade-off with coverage on the one hand and accuracy and complexity on the other hand. However, the projective algorithms do not work well on a few languages with a large percentage of non-projective sentences. It is thus desirable to find a method to deal with non-projectivity without sacrifice performance on projective sentences.

The projectivity assumption is also commonly imposed on the analysis of DLM in many previous studies, i.e., non-projective sentences are discarded in the experiments. In fact, non-projectivity is an equally if not more important factor of word order, and it should not be overlooked in the analysis. Therefore, we conduct experiments specifically on the non-projective sentences and show that the non-projectivity as a factor of word order is also strongly influenced by DLM.

1.2 Outline

The remainder of the dissertation is organized as follows:

> **Chapter 2** introduces the main concepts relevant to the thesis. We start with the framework of dependency grammar and the dichotomy of structural order and surface order, and the tasks of transforming from one to the other, namely parsing and linearization. We briefly introduce parsing methods, although the focus of this thesis is on linearization, since many concepts and methods that are originally

developed for parsing and adapted for linearization. We then describe several successful models for linearization. As our model is based on neural networks, we also introduce relevant models such as LSTM, Tree-LSTM, and Graph Neural Networks. Finally, we describe the Dependency Length Minimization hypothesis, which is a probable explanation of how the word order preferences in natural languages are shaped.

Chapter 3 describes our framework of the computational model for linearization, which consists of two main components, encoder and decoder. We present different methods to represent the dependency tree and different decoding algorithms to produce the surface order of the encoded dependency tree. We discuss the pros and cons of our decoder's inductive bias that restricts the surface order to be projective and present a transition-based model to reorder the sentence.

Chapter 4 describes the main experiments in this thesis. We start with the comparison of encoders and decoders to find the optimal combination for the linearization model. With the optimal linearization model, we then analyze its prediction errors from different angles. From the machine learning perspective, we quantify the extent of model errors and search errors to determine the performance bottleneck. From the linguistic perspective, we look for error patterns with respect to language families and dependency relations. We also conduct manual inspection into the error cases in a few languages and distinguish the word order variation from the actual linearization error.

Chapter 5 extends the linguistic analysis of word order in the previous chapter and explores a hypothesis of a universal factor that influences the word order, Dependency Length Minimization (DLM). We demonstrate the effect of DLM by designing a series of baselines and show that the dependency length would gradually decrease from complete random ordering to the actual ordering in the treebank data through several intermediate baselines. We explain each drop of dependency length with respect to a factor of word order and show the universality of DLM in almost all languages and all factors. Finally, we also explore the DLM hypothesis in projectivity, an often overlooked factor in word order.

Chapter 6 turns the focus back to the computational model. We apply the linearization model as a part of the pipeline for the shared tasks of surface realization. We describe other components in the pipeline as well as the data augmentation and ensembling methods for pushing the performance.

Chapter 7 concludes this thesis. We summarize the contribution in our work, which are mainly (1) developing an efficient and stable computational model for word order, (2) exploring the relationship between word order and dependency length, and (3) a special focus on non-projective trees. We also discuss our work in a broader perspective and envisage an interplay of linearization and parsing as syntax-based machine translation.

1.3 Publications

This main body of dissertation is based on the following peer-reviewed publications:

- Yu, X., Falenska, A., and Kuhn, J. (2019b). Dependency length minimization vs. word order constraints: An empirical study on 55 treebanks. In *Proceedings of the First Workshop on Quantitative Syntax (Quasy, SyntaxFest 2019)*, pages 89–97, Paris, France. Association for Computational Linguistics

  Chapter 5 is based on this publication. My co-author Agnieszka Falenska was responsible for visualization, Jonas Kuhn was responsible for writing the overview and providing advice.

- Yu, X., Falenska, A., Vu, N. T., and Kuhn, J. (2019c). Head-First Linearization with Tree-Structured Representation. In *Proceedings of the 12th International Conference on Natural Language Generation*, pages 279–289, Tokyo, Japan. Association for Computational Linguistics

  Chapter 3 is partly based on this publication. My co-author Agnieszka Falenska was responsible for visualization and calibrating baseline models, Ngoc Thang Vu and Jonas Kuhn had advisory roles.

- Yu, X., Falenska, A., Haid, M., Vu, N. T., and Kuhn, J. (2019a). IMSurReal: IMS at the Surface Realization Shared Task 2019. In *Proceedings of the 2nd Workshop on Multilingual Surface Realisation (MSR 2019)*, pages 50–58, Hong Kong, China. Association for Computational Linguistics

  Chapter 6 is partly based on this publication. My co-author Agnieszka Falenska was responsible for visualization and calibrating baseline models, Marina Haid conducted preliminary experiments, Ngoc Thang Vu and Jonas Kuhn had advisory roles.

- Yu, X., Tannert, S., Vu, N. T., and Kuhn, J. (2020a). Fast and accurate non-projective dependency tree linearization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1451–1462, Online. Association for Computational Linguistics

  Chapter 3 is partly based on this publication. My co-author Simon Tannert was responsible for visualization and conducting preliminary experiments, Ngoc Thang Vu and Jonas Kuhn had advisory roles.

- Yu, X., Tannert, S., Vu, N. T., and Kuhn, J. (2020b). IMSurReal too: IMS in the surface realization shared task 2020. In *Proceedings of the Third Workshop on Multilingual Surface Realisation*, pages 35–41, Barcelona, Spain (Online). Association for Computational Linguistics

  Chapter 6 is partly based on this publication. My co-author Simon Tannert was responsible for visualization and data augmentation, Ngoc Thang Vu and Jonas Kuhn had advisory roles.

The work in this dissertation has also been shaped by the experiences from the following publications during my doctoral studies. Although they are not a core part of the dissertation, they have been part of my project work and more or less influenced the final product.

- Yu, X. and Vu, N. T. (2017). Character composition model with convolutional neural networks for dependency parsing on morphologically rich languages. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 672–678, Vancouver, Canada. Association for Computational Linguistics

- Björkelund, A., Falenska, A., Yu, X., and Kuhn, J. (2017). IMS at the CoNLL 2017 UD shared task: CRFs and perceptrons meet neural networks. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 40–51, Vancouver, Canada. Association for Computational Linguistics

- Yu, X., Falenska, A., and Vu, N. T. (2017). A general-purpose tagger with convolutional neural networks. In *Proceedings of the First Workshop on Subword and Character Level Models in NLP*, pages 124–129, Copenhagen, Denmark. Association for Computational Linguistics

- Yu, X., Vu, N. T., and Kuhn, J. (2018). Approximate dynamic oracle for dependency parsing with reinforcement learning. In *Proceedings of the Second Workshop on Universal Dependencies (UDW 2018)*, pages 183–191, Brussels, Belgium. Association for Computational Linguistics

- Blohm, M., Jagfeld, G., Sood, E., Yu, X., and Vu, N. T. (2018). Comparing attention-based convolutional and recurrent neural networks: Success and limitations in machine reading comprehension. In *Proceedings of the 22nd Conference on Computational Natural Language Learning*, pages 108–118, Brussels, Belgium. Association for Computational Linguistics

- Yu, X., Vu, N. T., and Kuhn, J. (2019d). Learning the Dyck language with attention-based Seq2Seq models. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 138–146, Florence, Italy. Association for Computational Linguistics

- Yu, X., Vu, N. T., and Kuhn, J. (2020c). Ensemble self-training for low-resource languages: Grapheme-to-phoneme conversion and morphological inflection. In *Proceedings of the 17th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 70–78, Online. Association for Computational Linguistics

- Dönicke, T., Yu, X., and Kuhn, J. (2020b). Real-valued logics for typological universals: Framework and application. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 3990–4003, Barcelona, Spain (Online). International Committee on Computational Linguistics

- Dönicke, T., Yu, X., and Kuhn, J. (2020a). Identifying and handling cross-treebank inconsistencies in UD: A pilot study. In *Proceedings of the Fourth Workshop on Universal Dependencies (UDW 2020)*, pages 67–75, Barcelona, Spain (Online). Association for Computational Linguistics

Chapter 2

Background

2.1 Dependency Grammar

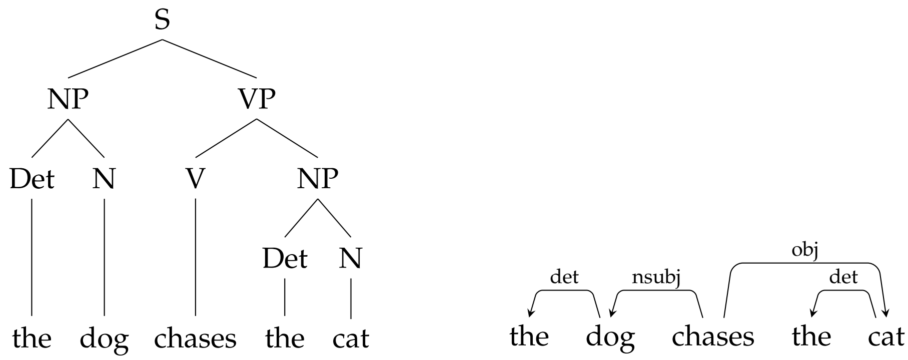
Dependency grammar is a grammar formalism representing the structure of a sentence by modeling the dependency relations among the words. The foundation of the modern dependency grammar is laid out by Tesnière (1959, 2015), followed by many theories with similar principles, such as the Word Grammar (Hudson, 1984), the Functional Generative Description (Sgall et al., 1986), the Meaning Text Theory (Mel'cuk et al., 1988), and Universal Dependencies (Nivre et al., 2016a).

The dependency grammar assumes the units of the structure are words, and they are connected by binary, asymmetric dependency relations, noted as $t_i \xrightarrow{l} t_j$, where t_i is the head, t_j is the dependent, and l is the dependency label between them, such as subject or object. If there is a directed path $t_i \rightarrow ... \rightarrow t_j$, then t_i is called the ancestor of t_j and t_j is the descendent of t_i, and we say t_i dominates t_j. In most theories, the dependency structure is a tree, i.e., there is a word as the root, which is the ancestor of all other words in the sentence. It satisfies the property that every word except for the root has exactly one head.

2.1.1 Dependency Grammar vs. Constituency Grammar

Another influential school of grammar formalism, the constituency grammar (also called phrase structure grammar) proposed by (Chomsky, 1957) represents the syntactical structure of a sentence with multiple levels of phrases. Figure 2.1 is an example that demonstrates the syntactical representation of a sentence in both formalisms.

There are several fundamental differences between these two grammar formalisms.



(a) constituency tree

(b) dependency tree

Figure 2.1: Examples of constituency tree and dependency tree.

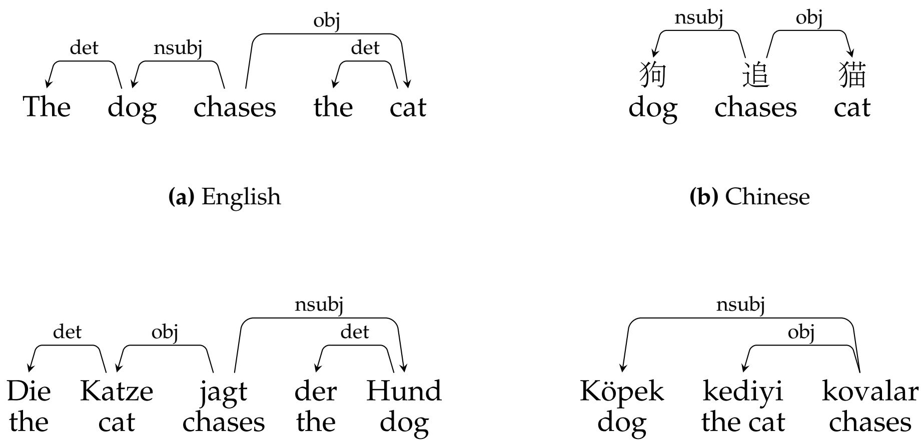
The most obvious contrast is that the constituency grammar introduces many intermediate nodes that are not present in the sentence's surface form. In contrast, the dependency grammar does not assume extra nodes apart from the words, and all nodes are directly linked to each other.

Another subtler difference is that the word order is coupled into the phrases but decoupled from the dependencies. In the example of constituency grammar, the order of the verb phrase has already determined that noun phrase *the mouse* follows the verb *chases*. However, in the dependency grammar, whether the verb *chases* precedes or follows its object *mouse*, is not encoded in its syntactic representation.

At first glance, the lack of surface order might be an inadequacy of the dependency grammar compared to the constituency grammar. However, in the context of multilingualism, it turns out to be a crucial advantage. The Universal Dependencies (UD) (Nivre et al., 2016b) is an initiative that maintains an extensive collection of treebanks aiming at a universal annotation guideline of dependency grammar across languages. It now contains more than 100 typologically diverse languages and is still rapidly expanding. Within the UD framework, the translation of a sentence in different languages can be represented as very similar dependency structures, although the word order might vary considerably.

Figure 2.2 illustrates the dependency trees of the sentence *the dog chases the cat* in four different languages. To amplify the contrast, the German example uses a less frequent but valid word order. There are three distinct orders of the verb and its arguments in these examples: SVO (subject-verb-object) for English and Chinese, OVS for German, and SOV for Turkish. Also, there are no explicit determiners for Chinese and Turkish.

However, all examples have a common core structure where the root of the sentence is *chases*, and it has two arguments, *dog* as the subject and *cat* as the object. The cross-lingual consistency of the (universal) dependency grammar is not only a plausible model of human language faculty but also practical for NLP applications. For example, it allows one to easily translate from one language to another by making the lexical substitution with proper morphological inflection, then arranging the word order of the target tree.



(a) English

(b) Chinese

(c) German

(d) Turkish

Figure 2.2: Dependency trees of the same meaning in different languages.

2.1.2 Projectivity

Projectivity is an important property to classify dependency trees. Intuitively, a tree is projective if its tokens are drawn in their linear order, and there are no dependency arcs crossing each other. For example, the trees in Figure 2.2 are all projective. If a tree has at least one non-projective arc, it is a non-projective tree. Figure 2.3 is an example of a non-projective tree, where the arc from *you* to *Mark* (marked in red) is non-projective.

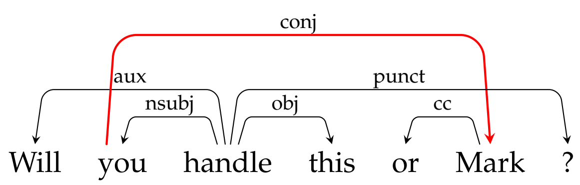


Figure 2.3: An example non-projective sentence.

Formally, an arc is called projective if the head of the arc dominates all intervening tokens between the head and the dependent, otherwise it is non-projective:

$$t_i \rightarrow t_j \text{ is projective} \iff \begin{cases} \forall_{t_i \prec t_k \prec t_j} \quad t_i \xrightarrow{*} t_k & \textbf{if } t_i \prec t_j \\ \forall_{t_j \prec t_k \prec t_i} \quad t_i \xrightarrow{*} t_k & \textbf{otherwise} \end{cases} \tag{2.1}$$

where $\prec$ denotes linear precedence.

2.1.3 Parsing and Linearization

Tesnière distinguishes the dependency structure and the surface word order as two different orders of the sentence, the **structural order** and the **linear order**. He argues that all the syntactic information of a sentence is contained in the structural order, while the linear order is a compromise to the physical world since time is one-dimensional and one has to utter the words linearly.

Since the linear order is independent of the structural order in dependency grammar, there is a need for a (computational) model that can transform one into the other. The task of transforming from linear order to structural order is **parsing**, and the task of transforming from structural order to linear order is **linearization**.

In natural languages (unlike programming languages), there is no perfect one-to-one mapping between the structural order and the linear order, making the task of parsing and linearization both difficult and exciting.

For parsing, the **syntactical ambiguity** causes different interpretations of the same sentence. For example, *I shot an elephant in my pajamas* could mean I was in the pajamas, or the elephant was in the pajamas. To resolve the ambiguity, one must rely on the semantics and world knowledge to understand that elephants are unlikely to wear pajamas.

For linearization, the **word order variation** results in the same syntactic structure being uttered in different ways. The previous example could be rearranged as: *In my pajamas, I shot an elephant*, where the emphasis is on the outfit instead of the event.

The word order variation is influenced by various factors at different linguistic levels, e.g., prosody (Zubizarreta, 1998), information structure (Hawkins, 1992; Lambrecht, 1996; Neeleman and Van de Koot, 2016), and as we will discuss in Chapter 5, the cognitive burden. Note that in the linearization model in this work, these factors are not explicitly encoded since we focus on the technical framework of data-driven linearization model. However, it is possible that the machine learning model could implicitly pick up the relevant cues to determine the word order.

2.2 Dependency Parsing Methods

Although this thesis does not directly work on dependency parsing, the parsing methods have profoundly influenced the development of linearization due to the similarity between these two tasks. Therefore we first briefly introduce some parsing methods, focusing on those directly adopted for linearization. For a complete introduction to dependency parsing algorithms, we refer to Kübler et al. (2009). Generally, there are two major paradigms of dependency parsing algorithms, graph-based and transition-based. Graph-based algorithms directly search for a tree in the fully-connected graphs, while transition-based algorithms derive the tree step by step through transitions.

2.2.1 Graph-based Parsing

Graph-based parsing algorithms (Eisner, 1996; McDonald et al., 2005) first construct a fully connected directed graph where every word is linked to every other word, a scoring function computes the scores of all the arcs in the graph, and a decoding algorithm finds a tree that maximizes the total score of the arcs in the tree, which is called the Maximum Spanning Tree (MST).

To score each potential arc, features are extracted from the head and the dependent of the arc, including their word forms, POS tags, their relative direction and distance, as well as other words around and between them as the context. These features are then used by the machine learning model, e.g., the perceptron (§2.4.1), to calculate a score.

Having the scores for all arcs in the graph, we can then use an MST algorithm to decode the tree. There are two commonly used decoding algorithms. Eisner's Algorithm (Eisner, 1996) is based on dynamic programming, which recursively breaks down the problem of finding the tree into finding the left and right subtrees and uses a chart to store the scores of the reoccuring subtrees to avoid repeated computation. Chu-Liu-Edmonds' Algorithm (Chu and Liu, 1965; Edmonds, 1967) used by McDonald et al. (2005) is a greedy algorithm with recursive refinement. It first selects the highest scoring incoming arc for each node, which might result in cycles (thus not a tree). It then resolves the cycles recursively until there is no more.

Both algorithms can find exact solutions efficiently, with the time complexity of $\mathcal{O}(n^3)$. A significant difference between the two is that Chu-Liu-Edmonds' Algorithms can natively produce non-projective trees, while Eisner's algorithm is restricted to projective trees. However, Eisner's algorithm can integrate more flexible higher-order features. Therefore, it is more popular for parsing mostly projective languages such as English.

Transition	Before			After		Precondition
left-arc	$\langle\sigma \mid t_i,$	$t_j \mid \beta, A\rangle$	$\Rightarrow$	$\langle\sigma,$	$t_j \mid \beta, A \cup \{\langle t_j, t_i\rangle\}\rangle$	$t_i \neq$ ROOT
right-arc	$\langle\sigma \mid t_i,$	$t_j \mid \beta, A\rangle$	$\Rightarrow$	$\langle\sigma,$	$t_i \mid \beta, A \cup \{\langle i, j\rangle\}\rangle$	
shift	$\langle\sigma,$	$t_i \mid \beta, A\rangle$	$\Rightarrow$	$\langle\sigma \mid t_i,$	$\beta, A\rangle$	

Table 2.1: Transitions of the ARCSTANDARD system.

2.2.2 Transition-based Parsing

Transition-based algorithms define a transition system that builds up the tree step by step through transitions that changes a parsing configuration to another configuration until reaching the terminal configuration, which contains the parse tree. Among many transition systems, we introduce the ARCSTANDARD system (Nivre, 2008) as an example, which has the same concepts such as configuration and transition that are also used in other transition systems.

A **configuration** is a triple $\langle\sigma, \beta, A\rangle$, where the stack σ contains partially processed tokens, the buffer β contains the unprocessed tokens in the sentence, and the set A contains the dependency arcs created so far. The initial configuration is $\langle[\text{ROOT}], [t_1, t_2, ..., t_n], \emptyset\rangle$, where the stack contains only the root token, the buffer contains all the tokens in the sentence in its order, and the arc set is empty. The terminal configuration is $\langle[\text{ROOT}], [], A\rangle$, where the stack again only contains the root token, the buffer is empty, and the A contains all the arcs that constitute the parse tree.

There are three **transitions** in the ARCSTANDARD system, *left-arc*, *right-arc*, and *shift* the effect of these transitions are described in Table 2.1. The *left-arc* transition creates an arc from the top of the stack i to the front of the buffer j and removes the token i; *right-arc* creates an arc from the front of the buffer j to the top of the stack i, removes the token i and moves j back to the buffer; *shift* moves the front of the buffer to the top of the stack.

2.2.3 Non-Projective Parsing

Among the previously described parsing algorithms, only the graph-based parser with Chu-Liu-Edmonds' algorithm can natively produce non-projective trees. For the projective parsers, several additional methods are developed as pre- or post-processing to remedy this restriction, and we introduce two methods here.

Nivre and Nilsson (2005) proposed the pseudo-projective parsing. The basic idea is to modify the target non-projective tree into a (pseudo-)projective one, by "lifting" the non-projective arcs until it becomes projective. To lift an arc is to change the head of the arc as the head of the head, and leave a trace in the dependency label. They can then use a

Transition	Before			After		Precondition
left-arc	$\langle\sigma \mid t_i,$	$t_j \mid \beta, A\rangle$	$\Rightarrow$	$\langle\sigma,$	$t_j \mid \beta, A \cup \{\langle t_j, t_i\rangle\}\rangle$	$t_i \neq$ ROOT
right-arc	$\langle\sigma \mid t_i,$	$t_j \mid \beta, A\rangle$	$\Rightarrow$	$\langle\sigma,$	$t_i \mid \beta, A \cup \{\langle i, j\rangle\}\rangle$	
shift	$\langle\sigma,$	$t_i \mid \beta, A\rangle$	$\Rightarrow$	$\langle\sigma \mid t_i,$	$\beta, A\rangle$	
swap	$\langle\sigma \mid t_i,$	$t_j \mid \beta, A\rangle$	$\Rightarrow$	$\langle\sigma,$	$t_j \mid t_i \mid \beta, A\rangle$	ROOT $\prec t_i \prec t_j$

Table 2.2: Transitions of the ARCSTANDARDSWAP system.

projective parsing algorithm to predict the pseudo-projective tree, and use the predicted label to recover the lifting thus producing a non-projective tree.

Nivre (2009) proposed another algorithm which modifies the projective ARCSTANDARD system algorithm by adding a new transition *swap*, which swaps the order between the top of the stack and the front of the buffer. The resulting system is noted as ARCSTANDARDSWAP. This transition essentially reorders the sentence such that the target parse tree could be projective in the new order. The transitions in the ARCSTANDARDSWAP are listed in Table 2.2, where the first three transitions are identical as in ARCSTANDARD.

The two methods above modify the non-projective trees in different ways: pseudo-projective parsing changes the tree structure by lifting the arcs; the ARCSTANDARDSWAP system changes the linear order of the sentence by swapping the words.

2.3 Linearization Methods

The research of computational models for linearization has its root in machine translation, which serves to find the order of words in the translated sentence (Brown et al., 1990; Brew, 1992). Earlier work mostly focuses on grammar-based approaches using different syntactic formalisms (Elhadad and Robin, 1992; Lavoie and Rainbow, 1997; Carroll et al., 1999). With the increasing availability of annotated treebanks, statistical methods gain popularity and become the dominant approach (Langkilde and Knight, 1998; Bangalore and Rambow, 2000; Filippova and Strube, 2009).

We now introduce a few statistical linearization models that are relevant to our work. We broadly divide the models into two categories based on the input format, syntax-free and syntax-based. Syntax-free linearization operates on a bag of words and does not assume any structure in them, while syntax-based linearization takes an unordered dependency tree (among other grammar formalisms) as input. Although the two categories differ a lot in the available information, many methods designed for one category can be adapted for the other category.

2.3.1 Syntax-free Linearization

Knight (1999) cast the task of linearization as a Traveling Salesman Problem (TSP). A TSP is a problem that takes a list of cities and the distance between each pair of cities and searches for a route that visits each city exactly once and returns to the origin. To formulate the word ordering problem as a TSP, each word is treated as a city, the log probability of a word pair in a bigram language model is treated as the distance, and the goal is to find a sequence of words with the maximum probability.

Much work has followed this formulation. Among others, Zaslavskiy et al. (2009) formulated the word ordering problem in phrase-based machine translation as a TSP and show that it achieves better performance and speed than beam search decoding with the same bigram language model. Horvat and Byrne (2014) explored higher-order n-gram language models for TSP-based word ordering, which transforms into a much larger TSP graph. These studies operate on a bag of words without syntax, which is a TSP graph of non-trivial size with little information about the internal structure. Much effort has been put into incorporating more powerful decoding algorithms such as Integer Programming (Germann et al., 2001) and Dynamic Programming (Tillmann and Ney, 2003).

Under the TSP formulation, the computational model behind linearization is a simple n-gram language model, which does not model long-distance relations. To address this problem, Schmaltz et al. (2016) proposed a linearization model based on a Long Short-Term Memory (LSTM) language model. They use a pre-trained language model, which can give the probability estimation for any suffix of a sentence, and use beam search to approximately search for the order that maximizes the overall probability.

2.3.2 Syntax-based Linearization

The methods mentioned above aim at linearizing a bag of words without any syntactical information. Although they could be used for dependency trees, the performance would be suboptimal. Syntax-based linearization typically uses discriminative models (in contrast to the generative models such as the language model) to decide the word order.

Bohnet et al. (2010) designed an algorithm to order the words with beam search. They extract features including the word form, part-of-speech, and dependency relations from the previous words and use a perceptron to calculate the scores for the next word. This is different from the approach based on the language model, which calculates the probability of the next word given the previous words only based on the statistics of co-occurrence of the words in a large corpus, The discriminative model can incorporate rich structural information from the input and make a more informed decision.

Another major property of their algorithm that benefits from the tree structure is that they do not order the full tree altogether. Instead, they divide the tree into subtrees (or *word order domains* in their terminology), where each subtree contains a head and its dependents. They then use beam search to order each subtree separately and finally combine the ordered subtrees into a full sentence. This is commonly referred to as the divide-and-conquer strategy since they divide a large problem (ordering the full tree) into several smaller subproblems (ordering each subtree), and by doing so, the search space is drastically reduced, thus increasing the likelihood of an approximate search algorithm to find the optimal solution.

A potential problem of this linearization algorithm is that by combining the ordered subtrees, it can only produce a projective sentence. Since the was initially designed to work on English data, which is predominantly projective, the inability to produce non-projective sentences did not pose a severe limitation to the model's performance. However, in order to apply the model to multilingual data, such limitation must be addressed. Bohnet et al. (2012) introduced a preprocessing model that first lifts the non-projective dependency arcs so that the correct order of the resulting dependency tree could be projective, although the tree is not the original one anymore. This method is inspired by the pseudo-projective parsing algorithm (Nivre and Nilsson, 2005).

Another line of work (Liu et al., 2015; Puduppully et al., 2016) is based on a transition system very similar to the ARCSTANDARD system for dependency parsing. They use a stack and a *set* to represents a configuration (in contrast to a stack and a *buffer* in a parsing configuration). There are three transitions, *shift* moves one word from the set onto the stack, *left-arc* and *right-arc* creates a dependency arc between the top two items on the stack and removes the dependent from the stack. If the dependency tree structure is not given (i.e., ordering a bag of words), this model performs parsing and linearization simultaneously, where *shift* decides the order of the words and *left-arc* and *right-arc* determine the dependency relations. If the dependency tree (or a partial tree) is given, then it is used as syntactic constraints to limit the possible transitions at every configuration. Since the transition system is based on the arc-standard system, it only produces projective sentences. Similar to Bohnet et al. (2010), a perceptron is used to score the transitions based on features extracted from the configuration. They also use beam search to explore multiple transition sequences.

2.4 Machine Learning Models

2.4.1 Perceptron

Traditional models for classification tasks in NLP are mostly based on linear models such as the perceptron (Rosenblatt, 1958; Collins, 2002), which makes the prediction based on features extracted from the instances. The features are extracted according to manually designed templates, such as "what is the lemma of the head", "what is the POS tag of the dependent". The extracted features are then "the lemma of the head is *chases*", "the POS tag of the dependent is NOUN". All the features found in the training set are assigned an index and stored in a feature mapping. In this way, each instance can be mapped to an input vector x composed of 0s and 1s, where the values at the positions that correspond to the activated features are set to 1, and the rest are set to 0.

The computation in a perceptron model is:

$$y = W \cdot x + b \tag{2.2}$$

where the weight matrix W and the bias vector b are the model parameters, x is the input feature vector, and y is the output vector that holds the score for each class. And the prediction of the model is the class with the highest score in y.

To increase the linear perceptron model's expressive power, feature combination is often needed, where several elemental features are merged as one. It is not trivial to design a proper set of feature templates. The model could suffer from **underfitting** if the features are not expressive enough, or suffer from **overfitting** if the features are too specific that only matches certain training instances and cannot generalize. For this reason, much effort has been focused on feature engineering, i.e., exploring informative elemental features and feature combinations that strike a balance between underfitting and overfitting.

2.4.2 Multi-Layer Perceptron

The emergence of neural networks has brought profound influence into the NLP field. One of the most basic neural models is the Multi-layer Perceptron (MLP), which is a series of perceptrons with non-linear functions (e.g., the sigmoid function) applied between the layers. This introduces non-linearity into the computation, thus frees the need for manual feature combinations.

Following is an example of an MLP with two hidden-layers:

$$h_1 = \sigma(W_1 \cdot x + b_1) \tag{2.3}$$

$$h_2 = \sigma(W_2 \cdot h_1 + b_2) \tag{2.4}$$

$$y = W_3 + h_2 + b_3 \tag{2.5}$$

where x and y are the input and output vectors (same as in perceptron), W_1, W_2, W_3 are weight matrices, and b_1, b_2, and b_3 are bias vectors, h_1 and $h2$ are the intermediate results from each layer of perceptron, and $\sigma(\cdot)$ is the non-linear function, for example the sigmoid function:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.6}$$

For simplicity, we use the following abstract notation to refer to the MLP:

$$y = MLP(x) \tag{2.7}$$

2.4.3 Distributed Representation

Arguably the most important concept of deep learning models in NLP is the distributed representation, also known as word embeddings when applied to represent words (Bengio et al., 2003; Mikolov et al., 2013). In the traditional perceptron model, the input is a one-hot vector, i.e., a vector with either 0 or 1, where the value on the n-th position being 1 means the n-th feature is activated. An expressive model typically has millions of features, which means the input vector and the model parameters would have millions of dimensions.

The distributed representation, in contrast, uses a much smaller, dense vector to represent the features (words), where the values of the vector are part of the model parameters that are trained together with the rest of the model. The distributed representation not only has the advantage of compactness, more importantly, it also captures the syntactic and semantic similarity between the words, which is obtained from the context during the training.

The distributed representation is initially used to represent words since it originates in language models, where words are the only observable features. It is soon adopted to represent any categorical features, such as POS tags, dependency relations (Chen and Manning, 2014).

2.4.4 Recurrent Neural Networks

Another advantage of deep learning models is their ability to model long-distance contexts. The Recurrent Neural Network (RNN) is designed by Elman (1990) to model sequences using a memory cell that recurrently updates itself with a sequence of inputs.

Suppose we have a sequence of vectors as input $[x_0, x_1, ..., x_n]$, where each x_i represents one element in the sequence. We can apply the RNN to obtain a sequence of output vectors $[y_0, y_1, ..., y_n]$, where each y_i also represents each element in the sequence, and it now takes its previous context into account.

The model maintains memory cell that keeps the information of the sequence that it reads so far, and at each step, it takes the current input, updates the memory, and outputs a vector as the output, as follows:

$$h_t = \sigma_h(W_h \cdot x_t + U_h \cdot h_{t-1} + b_h) \tag{2.8}$$

$$y_t = \sigma_y(W_y \cdot h_t + b_y) \tag{2.9}$$

where x_t is the input vector at time step t, [W_h, U_h, b_h] are parameters to calculate the hidden state h_t, [W_y, b_y] are parameters to calculate the output y_t, $\sigma_h(\cdot)$ and $\sigma_y(\cdot)$ are non-linear activation functions.

The vanilla RNN has the problem of vanishing or exploding gradient due to the depth of the layers that prevent stable and effective training. Hochreiter and Schmidhuber (1997) proposed a more stable variant, the Long Short-Term Memory (LSTM) architecture, that addresses this problem.

At each time step with the input x_t, the computation in of the LSTM is as follows:

$$f_t = \sigma(W_f \cdot x_t + U_f \cdot h_{t-1} + b_f) \tag{2.10}$$

$$i_t = \sigma(W_i \cdot x_t + U_i \cdot h_{t-1} + b_i) \tag{2.11}$$

$$o_t = \sigma(W_o \cdot x_t + U_o \cdot h_{t-1} + b_o) \tag{2.12}$$

$$u_t = \tanh(W_u \cdot x_t + U_u \cdot h_{t-1} + b_u) \tag{2.13}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot u_t \tag{2.14}$$

$$h_t = o_t \odot \tanh(c_t) \tag{2.15}$$

where f_t is the forget gate, i_t is the the input gate, and o_t is the output gate, all three vectors are calculated from the current input x_t, the previous output h_{t-1} with the corresponding parameters W, U, and b. These gating vectors all have values between 0 and 1, as a result of the sigmoid activation function, they are used to manipulate the transformed information of the current step u_t through element-wise multiplication $\odot$ to form the current memory c_t and the current output h_t.

The LSTM only models the sequence in one direction, which means the output at a time step is not aware of information from the future time steps. To better process the sequential information, Graves and Schmidhuber (2005) proposed the bidirectional LSTM, which uses two LSTMs that run from opposite directions, and the representation of each element in the sequence is simply the concatenation of the outputs of both LSTMs.

We use the following abstract notations for the forward and backward LSTMs:

$$y_{1:n} = \overrightarrow{\text{LSTM}}(x_{1:n}) \tag{2.16}$$

$$y_{1:n} = \overleftarrow{\text{LSTM}}(x_{1:n}) \tag{2.17}$$

where $x_{1:n}$ is a sequence of input vectors and $y_{1:n}$ is a sequence of output vectors.

2.4.5 Sequence-to-Sequence Model and Attention Mechanism

The Sequence-to-Sequence (Seq2Seq) architecture (Cho et al., 2014; Sutskever et al., 2014) is a widely successful model that takes a sequence as input and produces another sequence as output. It is typically based on the combination of two RNNs, one RNN as the encoder that reads the input sequence, the other as the decoder that incrementally generates the elements of the output sequence.

The encoder reads all elements in the input sequence. Its final output summarizes all the information in the input and is used by the decoder to generate the output. This introduces a problem, namely, it has to compress all the information in a sequence into one vector, which would cause information loss, especially for longer sequences, thus hurting the quality of the generated output.

The attention mechanism (Graves et al., 2014; Bahdanau et al., 2014; Luong et al., 2015) is designed to address this problem. It is motivated by the fact that humans pay attention to different elements in the input sequence at different times. For example, in machine translation, when generating each word in the output, different input words become the focus. In a Seq2Seq model with attention, the current decoder state calculates its attention over all encoder outputs at every time step. The attention is a vector with values ranging between 0 and 1 and summing up to 1. It is used to obtain a weighted average of all the encoder states, which is then used by the decoder to generate the output for the current time step.

Generally, the attention score on one encoder state is calculated as follows:

$$s_{t,i} = f(y_t, x_i) \tag{2.18}$$

where $f(y_t, x_i)$ is a function $\mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ that takes the decoder state y_t and the encoder state x_i. There are several different attention models, which only differ in the function $f(y_t, x_i)$, and we refer to Hu (2019) for a comprehensive survey. The function that we use is based on Vaswani et al. (2017):

$$f(y_t, x_i) = \frac{x_i^\top \cdot y_t}{\sqrt{d}} \tag{2.19}$$

where d is the dimension of the encoder state.

The attention score is then normalized by the softmax function:

$$\alpha_{t,i} = \frac{\exp(s_{t,i})}{\sum_{j=1}^{n} \exp(s_{t,j})} \tag{2.20}$$

Having the attention score $\alpha_{t,i}$, we can then calculate the weighted average of the encoder state, which focuses on different position of the input at each decoding step:

$$c_t = \sum_{i=1}^{n} \alpha_{t,i} \cdot x_i \tag{2.21}$$

We use the following abstract notation for the attention mechanism:

$$\alpha_{t,1:n} = \text{attention}(y_t, x_{1:n}) \tag{2.22}$$

where $x_{1:n}$ are the encoder states for the elements in the input, y_t is the decoder state at the t-th step.

In the attention-based Seq2Seq models, the attention mechanism is used to calculate the weights to average the representations of the input vectors, and the output is generated from the vocabulary. Vinyals et al. (2015b) proposed the pointer network, which can be viewed as a simplified version of the attention-based Seq2Seq model. It also calculates the attention between the current decoder state and all encoder states, but instead of generating an output, it simply picks the input item with the highest attention as the output. The pointer network model is suitable for the tasks where all elements in the output are contained in the input, for example, the linearization task.

2.4.6 Tree-LSTM

Tai et al. (2015) proposed the Tree-LSTM to model tree structures, which is a generalization of the sequential LSTM. We introduce the Child-sum variant here, as it is suitable to model an unordered dependency tree. Unlike LSTM that propagates information from the preceding node to the following node in a sequence, Tree-LSTM propagates the information from the dependents to their head in a dependency tree. Suppose we calculate the node j, whose dependents are $k \in C(j)$, the computation of the Tree-LSTM is:

$$\tilde{h}_j = \sum_{k \in C(j)} h_k \tag{2.23}$$

$$f_{jk} = \sigma(W_f \cdot x_j + U_f \cdot h_k + b_f) \tag{2.24}$$

$$i_j = \sigma(W_i \cdot x_j + U_i \cdot \tilde{h}_j + b_i) \tag{2.25}$$

$$o_j = \sigma(W_o \cdot x_j + U_o \cdot \tilde{h}_j + b_o) \tag{2.26}$$

$$u_j = \tanh(W_u \cdot x_j + U_u \cdot \tilde{h}_j + b_u) \tag{2.27}$$

$$c_j = i_j \odot u_j + \sum_{k \in C(j)} f_{jk} \odot c_k \tag{2.28}$$

$$h_j = o_j \odot \tanh(c_j) \tag{2.29}$$

We first sum up the output of all the dependents of node j to obtain $\tilde{h}_j$ that summarizes the information of the dependents, then calculate the forget gate of each dependent f_{jk}, the input gate i_j and the output gate o_j for the current node j. The memory cell of each dependent is controlled by the corresponding forget gate, the transformed information u_j is controlled by the input gate, and their sum forms the memory cell of the current node, and finally outputs h_j through the output gate.

The Tree-LSTM is improved by adding the attention mechanism to the hidden states (Zhou et al., 2016), so that each dependent influences the head representation to different degrees. Miwa and Bansal (2016) proposed a bidirectional extension that traverses the tree in both bottom-up and top-down directions to access all the tokens in the tree.

We use the following abstract notations for the bottom-up and top-down Tree-LSTMs:

$$y^{\uparrow}_{1:n} = \text{Tree-LSTM}^{\uparrow}(x_{1:n}) \tag{2.30}$$

$$y^{\downarrow}_{1:n} = \text{Tree-LSTM}^{\downarrow}(x_{1:n}) \tag{2.31}$$

where $x_{1:n}$ are the input vectors of the nodes in the tree, and $y^{\uparrow}_{1:n}$ and $y^{\downarrow}_{1:n}$ are the output vectors of the bottom-up and top-down Tree-LSTMs, respectively.

2.4.7 Graph Neural Networks

A tree is a particular case of a graph, and the general graph structure has much broader application. Therefore, there has been a lot of work focusing on modeling graph structures with Graph Neural Networks (GNNs); we refer the reader to Wu et al. (2020) for a comprehensive survey. Here, we introduce a variant we use in our experiments, the Graph Attention Network (GAT) by Veličković et al. (2017), which is improved upon the Graph Convolutional Network (GCN) by Kipf and Welling (2016).

Unlike the sequence or tree structure, there is no beginning or ending in a graph; therefore, we cannot propagate the information in one go as in the sequential LSTM and Tree-LSTM. Instead, every node in a connected graph has neighbors, and the GNN propagates the information of every node to its neighbors simultaneously and repeats it n times to propagate further. Each propagation uses a separate layer of GNN, and it can thus only send messages to a limited number of hops way, which is a compromise to process more complex structure compared to the Sequential or Tree-LSTM.

In the basic GCN model, the representation of a node i at the $(l+1)$-th layer is based on the representations of its neighbors in the l-th layer as follows:

$$z_j^{(l)} = W^{(l)} \cdot h_j^{(l)} \tag{2.32}$$

$$\alpha_{i,j} = \frac{1}{\sqrt{|\mathcal{N}(i)| \cdot |\mathcal{N}(j)|}} \tag{2.33}$$

$$h_i^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}(i)} \alpha_{i,j} \cdot z^{(l)}\right) \tag{2.34}$$

where $\mathcal{N}(i)$ are the direct neighbors of i, $z_j^{(l)}$ is the transformed representation of a neighbor node j at the previous layer, $\alpha_{i,j}$ is the normalization term to calculate the average.

When computing the representation of a node, GCN averages the information from its neighbors, while GAT takes into account the importance of each neighbor by calculating the attention from the node to its neighbors, by replacing Equation (2.33) with:

$$\alpha_{i,j} = \text{attention}(h_i^{(l)}, h_{1:n}^{(l)})_j \tag{2.35}$$

We use the following abstract notation for the GAT:

$$y_{1:n} = \text{GAT}(x_{1:n}) \tag{2.36}$$

where $x_{1:n}$ and $y_{1:n}$ are the input and output vectors for all the nodes in the graph.

2.5 Dependency Length Minimization

The recent development of comparable treebanks for a wide range of languages across the typological spectrum, particularly the UD (Nivre et al., 2016b), has made it possible to address some long-standing hypotheses regarding a functional explanation of linguistic universals on word orders. Several recent studies (Liu, 2008; Futrell et al., 2015a) have used evidence from treebanks across languages to address what is arguably the most prominent hypothesis of a functionally motivated universal constraint, the Dependency Length Minimization (DLM) hypothesis, which can be traced back to Behaghel (1932). The DLM hypothesis states that we tend to arrange the words to minimize the overall lengths of the dependency arcs in the linear order to reduce the cognitive load of understanding (parsing) the sentence. This effect of DLM presents in the utterances where multiple alternative word orders are valid; it also influences the preferences of word order in the long-term development of a language.

Liu (2008) performed the first cross-treebank study to compare the actual length of dependencies, for 20 languages, against the length that results when the dependency structures are linearized in random ways. The results indicate that languages indeed tend to minimize the dependency distance. Futrell et al. (2015a) presented a recent expansion of this type of treebank study to a set of 37 languages (which they argue to be the first comprehensive analysis that covers a broad range of typologically diverse languages), presenting comparisons of the real dependency trees from the treebank with random re-orderings of the same dependency structures. The analysis shows that the real DL is indeed significantly shorter than chance across all 37 analyzed languages. This result corroborates findings from a broad range of empirical studies that are typologically less comprehensive (Gildea and Temperley, 2010; Gulordava and Merlo, 2015; Gulordava et al., 2015). This type of cross-treebank study has prompted a fair number of expansions and discussion regarding the typological implications (e.g., Jiang and Liu (2015); Chen and Gerdes (2018); Temperley and Gildea (2018)).

The studies mentioned above (Liu, 2008; Futrell et al., 2015a; Gulordava, 2018) have only focused on the projective sentences and ignored the non-projective sentences, which constitute a substantial proportion in many languages with relatively free word order. A series of studies (Ferrer-i-Cancho, 2006; Ferrer-i-Cancho and Gómez-Rodríguez, 2016; Gómez-Rodríguez and Ferrer-i-Cancho, 2017) focused on the relation between crossing arcs and DL. They concluded that the scarcity of non-projective sentences in natural languages is a direct outcome of DLM since the crossing arcs generally would increase the DL of an otherwise projective sentence.

Chapter 3

Dependency Tree Linearization

The task of linearization is to arrange the nodes in a graph structure into a sequence. In our context, the input is an unordered dependency tree, and the output is a sentence. Linearization on its own may seem somewhat artificial since it is uncommon to have the dependency tree structure of a sentence while the order is unknown. However, this could be an essential step for many real-world tasks. For example, in the deep track of several editions of the surface realization shared task (Belz et al., 2011; Mille et al., 2018, 2019, 2020), the input is an deep-syntactic structure which contains only content words connected by predicate-argument relations (Meyers et al., 2004; Palmer et al., 2005).

Figure 3.1 illustrates our complete pipeline for the surface realization shared task for both shallow and deep tracks. It consists of five modules: (1) **linearization** aims to find the surface order of a tree; (2) **completion** is the process of transforming the deep tree into a shallow tree by generating the absent function words; (3) **inflection** is the task of generating the word form based on the lemma and the syntactic and morphological information; (4) **contraction** applies for a few languages where several tokens are contracted as one; (5) **detokenization** turns a list of words into text with proper spacing between the words and punctuation marks. The code for pipeline with all the modules is available at `https://github.com/EggplantElf/IMSurReal`.

This chapter focuses on the first module in the pipeline, dependency tree linearization. This module aims to find the optimal order of an unordered dependency tree, be it shallow or deep; it takes a tree as input and produces a sequence of tokens from the tree. A typical linearization model includes two major components: encoding and decoding, where the tokens are encoded with the tree information and then decoded into a sequence that forms a natural language sentence.

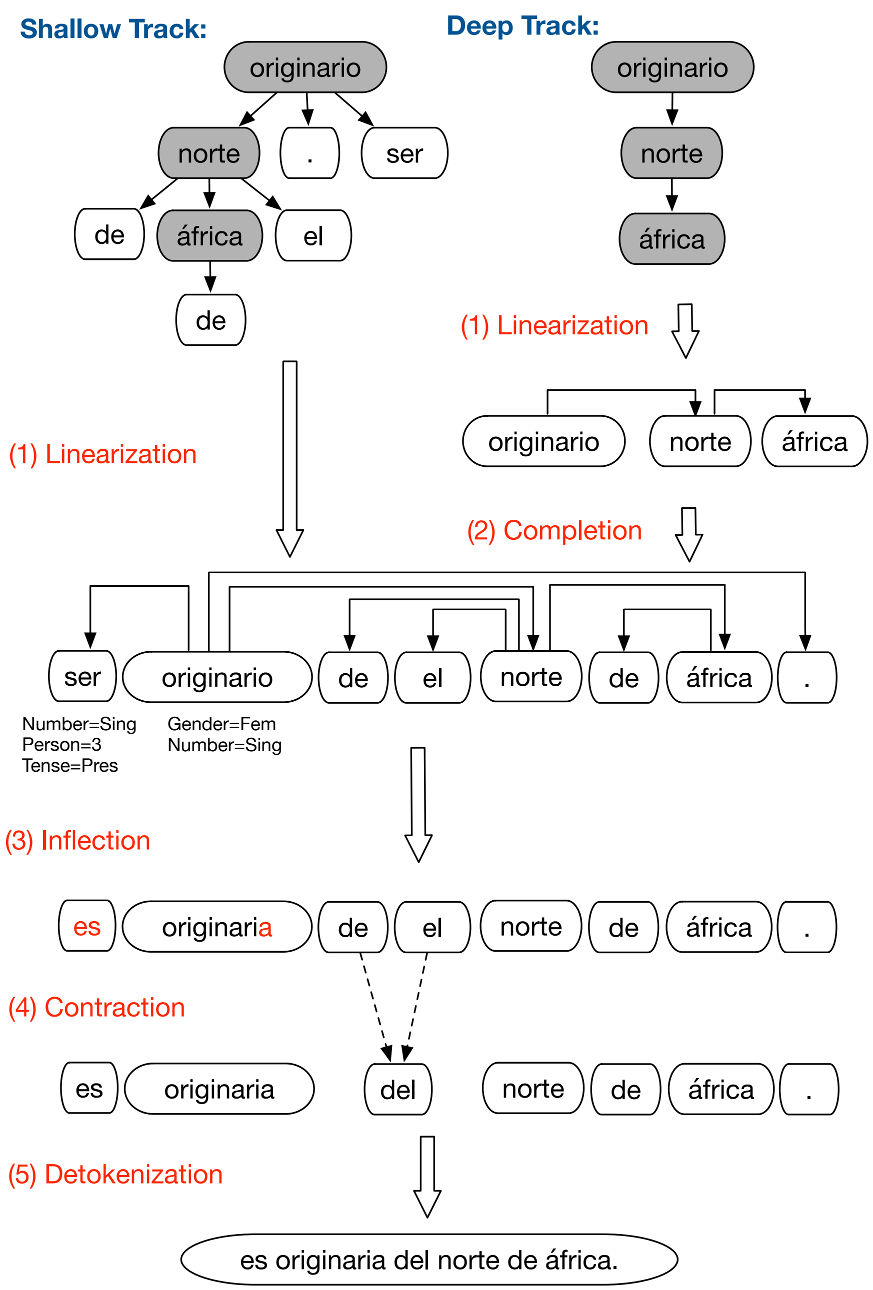


Figure 3.1: A full pipeline for the surface realization shared task, including five modules: (1) linearization, (2) completion, (3) inflection, (4) contraction, and (5) detokenization.

3.1 Encoding

We explore three models to encode the input unordered dependency tree: sequential LSTM, Tree-LSTM, and GAT. These three models are designed for structures with increasing generality, i.e., sequence, tree, and graph.

3.1.1 Token

Before modeling the structure, we first obtain the representation for each token in the tree, which serves as the common ground for all the structural encoders. Table 3.1 shows an example sentence from the treebank in the CoNLL-U format. Each row contains the information about a token in the sentence, including the token index, word form, lemma, Universal Part-of-Speech (UPOS), Language-specific Part-of-Speech (XPOS), a list of morphological features, the index of the head, and the dependency relation.

ID	Word	Lemma	UPOS	XPOS	Morphological Features	Head	Label	_	_
1	From	from	ADP	IN	_	3	case	_	_
2	the	the	DET	DT	Definite=Def|PronType=Art	3	det	_	_
3	AP	AP	PROPN	NNP	Number=Sing	4	obl	_	_
4	comes	come	VERB	VBZ	Mood=Ind|Number=Sing|Person=3|...	0	root	_	_
5	this	this	DET	DT	Number=Sing|PronType=Dem	6	det	_	_
6	story	story	NOUN	NN	Number=Sing	4	nsubj	_	_
7	:	:	PUNCT	:	_	4	punct	_	_

Table 3.1: An example sentence in the CoNLL-U format.

Among these entries, the token index and head index indicate the correct order of the words in the sentence, they are available for training but scrambled for testing. The word form is the target for morphological inflection, which is also not available for linearization. Therefore, we use the lemma, UPOS, morphological features, and dependency label as features to encode the token; XPOS is left out since they contain different information in different treebanks thus making a cross-lingual comparison more difficult.

We run a bidirectional LSTM on the embeddings of the characters in the lemma (denoted as $\mathbf{c}_{1:p}$) and a bidirectional LSTM for the morphological tags (denoted as $\mathbf{m}_{1:q}$).

$$\mathbf{v}_i^{(char)} = \overrightarrow{\text{LSTM}}^{char}(\mathbf{c}_{1:p})_p + \overleftarrow{\text{LSTM}}^{char}(\mathbf{c}_{1:p})_1 \tag{3.1}$$

$$\mathbf{v}_i^{(morph)} = \overrightarrow{\text{LSTM}}^{morph}(\mathbf{m}_{1:q})_q + \overleftarrow{\text{LSTM}}^{morph}(\mathbf{m}_{1:q})_1 \tag{3.2}$$

The LSTM output vectors for characters and morphological tags are then summed together with the embeddings of other features, i.e., lemma, UPOS, and dependency relation, to form the token-level representation.

$$\mathbf{v}_i^{\circ} = \mathbf{e}_i^{(lemma)} + \mathbf{e}_i^{(upos)} + \mathbf{e}_i^{(dep)} + \mathbf{v}_i^{(char)} + \mathbf{v}_i^{(morph)} \tag{3.3}$$

Since the training data are generally relatively small (thousands of sentences), we apply word dropout during training to improve the robustness against unknown words. Concretely, During training, we set the lemma representation as a zero vector with a probability of 1%, so the model is forced to rely on other features to make the prediction.

The token-level encoding can also be viewed as the simplest way to encode the tree, ignoring the complete syntactic information and modeling it as a bag of words. Such encoding generally leads to poor performance, but it could serve as a baseline in the ablation study to illustrate the importance of syntactic information.

3.1.2 Sequence

Since the Sequence-to-Sequence (Seq2Seq) models gained popularity with their success in the machine translation task, they have also been widely adopted in many other tasks, even when the input is not a sequence in nature, including the linearization task (Konstas et al., 2017; Elder and Hokamp, 2018; Elder et al., 2020).

To use the Seq2Seq model to represent the tree structure, we first need to transform the tree structure to a sequence, which still include all the information about the tree. We denote this process as **serialization**. Note that the term appears quite similar to linearization, but they have very different meanings in our context. Serialization as a pre-processing step flattens the input unordered tree into a sequence with a depth-first, pre-order traversal, where the head is visited first, and the dependents are visited in arbitrary order. Linearization, in contrast, is the task the model is trying to solve, where the order of the output sequence is the target.

We follow Konstas et al. (2017) to use scoping brackets to establish an unambiguous mapping between the tree and the sequence, so that the tree information is not lost in the serialization. To improve the model's robustness to random serialization, we use a new random sequence every time to serialize the same tree during training.

Figure 3.2 illustrates different ways to represent an input dependency tree as a sequence. (1) and (2) are two different random serialization of the tree with the scoping brackets; (3) is a random order traversal of the tree without scoping brackets, which does not fully encode the tree information since we cannot unambiguously reconstruct the tree from the sequence; (4) is a completely random sequence, i.e., it is not necessarily a traversal of the tree in any order; thus it completely loses the tree information.

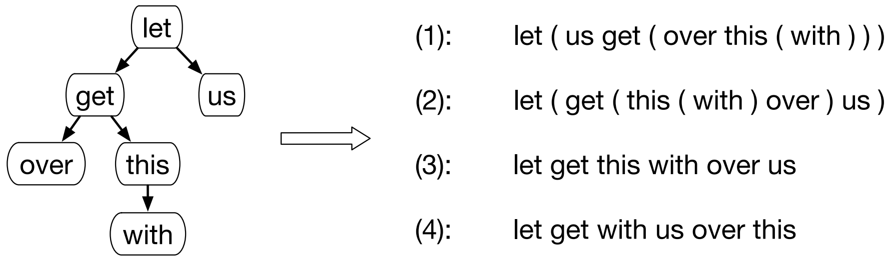


Figure 3.2: Different ways to serialize a dependency tree as a sequence.

Having transformed the tree into a sequence, we can then use a bidirectional LSTM to encode the sequence from left to right and from right to left. This way, the tokens in the sequence are encoded with a proxy of the tree information.

It is worth noting that, although the tree can be unambiguously transformed into a sequence, the LSTM model does not necessarily have the ability to understand the tree structure behind it. We have shown in Yu et al. (2019d) that the LSTM can recognize the tree structure to some extent, but it cannot truly generalize when the model is tested on a larger tree than what it is trained on. However, in this task, the training and testing sentences are generally of similar sizes; therefore, the limitation of generalization does not impose a severe problem.

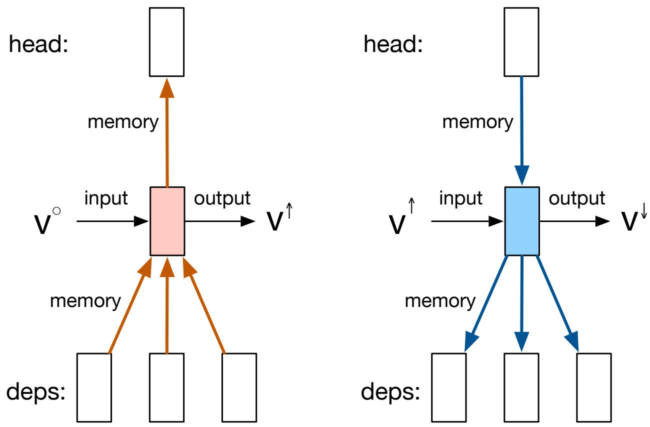
3.1.3 Tree-LSTM

To represent the tree nodes with better generalization ability, we adopt the Tree-LSTM model (Tai et al., 2015) to explicitly encode the dependency tree. The original Tree-LSTM propagates information only in the bottom-up direction, i.e., from the dependents to the heads, which does not encode all hierarchical information in each node since there is no information flow from the ancestor to the descendants. We therefore apply a top-down pass to propagate information from the head to the dependents as in Miwa and Bansal (2016). Since each dependent has only one head, unlike the bottom-up pass, we could use a standard sequential LSTM to encode the paths from the root to each leaf node. For each node, we feed its bottom-up vector $\mathbf{v}^{\uparrow}$ into the hidden state of its head to obtain the hidden state for the current node, and the output is the top-down vector $\mathbf{v}^{\downarrow}$. Miwa and Bansal (2016) perform the two passes independently, i.e., both LSTMs take $\mathbf{v}^{\circ}$ as input and produce $\mathbf{v}^{\uparrow}$ and $\mathbf{v}^{\downarrow}$ as outputs, similar to the standard sequential bidirectional

LSTM (Graves and Schmidhuber, 2005). However, two independent passes can not pass the information of all tokens to all other tokens in the tree. Since each token only gets information from its ancestors and descendants, it is thus unaware of its siblings, which is a crucial information for linearization.

Therefore, our model performs the bottom-up pass first and uses its output $\mathbf{v}^{\uparrow}$ as the input for the top-down pass to obtain $\mathbf{v}^{\downarrow}$, as illustrated in Figure 3.3. The final representation is the sum of the token-level vector and the two Tree-LSTM outputs.

$$\mathbf{v}_i^{\uparrow} = \text{Tree-LSTM}^{\uparrow}(\mathbf{v}_{0:n}^{\circ})_i \tag{3.4}$$
$$\mathbf{v}_i^{\downarrow} = \text{Tree-LSTM}^{\downarrow}(\mathbf{v}_{0:n}^{\uparrow})_i \tag{3.5}$$
$$\mathbf{x}_i = \mathbf{v}_i^{\circ} + \mathbf{v}_i^{\uparrow} + \mathbf{v}_i^{\downarrow} \tag{3.6}$$



(a) Bottom-up LSTM. **(b)** Top-down LSTM.

Figure 3.3: A schematic illustration of the bidirectional Tree-LSTM.

In this way, all tokens in the tree can be accessed by other tokens since any two tokens have a common ancestor, and the information of one token can be first passed up to the common ancestor then down to the other token. Figure 3.4 illustrates the information flow of our bidirectional model in a dependency tree, where the red dotted arrows indicate the bottom-up pass, and the blue dashed arrows indicate the top-down pass. We highlight how node 8 influences node 4. Its representation $\mathbf{v}_8^{\circ}$ is first propagated up to the lowest common ancestor $\mathbf{v}_2^{\uparrow}$, then goes down to $\mathbf{v}_4^{\downarrow}$.

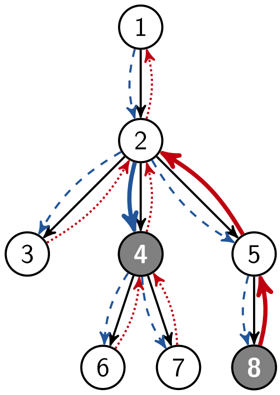


Figure 3.4: An illustration of the information flow in the encoder, where the red dotted arrows represent the bottom-up pass and the blue dashed arrows represent the top-down pass. The solid arrows illustrate the information flow from node 8 to node 4.

3.1.4 Graph Neural Networks

Graph neural networks are typically used to encode graph structures beyond trees. Nevertheless, we apply it to encode the input dependency tree. We implement the Graph Attention Network (GAT) model described in §2.4.7. Through preliminary experiments, we set the number of GAT layers to 3, i.e., each node is aware of its neighbors three hops away. This is a compromise between the model's expressiveness and parsimony: too few layers would cause the node to have insufficient context information, while too many layers would bloat the model parameters and cause overfitting.

Let us take the example in Figure 3.4 to compare Tree-LSTM and GAT. The Tree-LSTM has two sets of parameters, i.e., the bottom-up and the top-down submodule. When passing the information from node 8 to node 4, the bottom-up submodule is used twice for the information to reach node 2, and the top-down submodule is used once to pass it down to node 4. In contrast, the GAT in our setting has three sets of parameters, i.e., the three layers, and they can pass the information three steps away, just enough from node 8 to node 4, and each step involves a GAT layer. However, it is not possible to pass the information further to node 7. Therefore, the GAT has a limited scope of structural context, and its node representation may not be aware of the full graph.

3.2 Divide-and-Conquer Strategy

After encoding the tokens, we could directly generate the sequence of the tree with a decoder. However, it is not trivial to order decode long sentences, since the total search space is the factorial of the number of tokens. The divide-and-conquer strategy is commonly used to break down a large problem into several smaller problems and solve them separately in order to reduce the problem's complexity. Following Bohnet et al. (2010), we employ this strategy by recursively traversing the tree and solve the linearization task for each local subtree, where each subtree consists of a head and its direct dependents.

After obtaining the subtrees' local orders (by the decoder in §3.3), there are multiple possible ways to combine them to obtain the total order. Figure 3.5 gives an example, where (a) and (b) are the linearized subtrees, and (c), (d), and (e) are three different full linearized trees that respect the subtrees' local order.

Out of all possibilities, there is only one projective tree, i.e., without crossing arc in the tree, which is (c) in the example. Therefore, we make a projective assumption on the output to complete the divide-and-conquer process. This assumption turns out to be a practical inductive bias since the majority of sentences in most languages are projective, or contain very few non-projective arcs, see the statistics in Table 4.1.

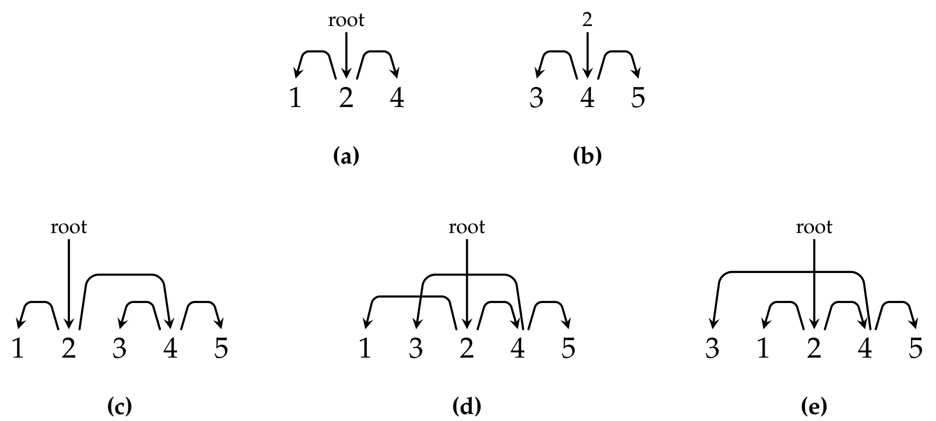


Figure 3.5: Multiple possible full tree orders from the same subtree orders, where (a) and (b) are ordered subtrees, (c), (d), and (e) are possible full ordered trees, among which only (c) is projective.

Whether to employ the divide-and-conquer strategy is a trade-off between the decoding efficiency and the coverage of exactly correct solutions. On the one hand, the strategy massively reduces the search space; on the other hand, it is impossible to produce a small fraction of correct trees since they lie in the reduced search space.

3.3 Decoding

We experiment with three types of decoding algorithm: incremental decoding, graph-based decoding, and transition-based decoding. In the first method, we add tokens incrementally to build partial sequences and use beam search to explore several hypotheses. In the second method, we treat the task as a Traveling Salesman Problem (TSP) and decode the sequence with a TSP solver. In the third method, we design a transition system to swap the tokens in the initial sequence step by step until reaching the final word order.

3.3.1 Incremental Decoding

Left-to-Right Generation with Beam Search

We start with the incremental decoding method, largely following Bohnet et al. (2010). The decoder takes a set of tokens as input, and the sequence is initially empty. At each decoding step, we select one token from the set of candidate tokens and add it to the end of the partial sequence until the candidate set is empty. To address the error propagation problem in greedy search, the decoder applies beam search to extend the sequence. At each step, we keep at most k partial sequences as the current hypotheses.

We use an LSTM to represent the partial sequences and adopt the pointer network (Vinyals et al., 2015b) to score the new expansions. At each step, we calculate the attention score between the LSTM output of the current partial sequence and the remaining tokens that are "pointed" to. The unnormalized attention score (without softmax) is the incremental score for adding such token.

Algorithm 1 describes the left-to-right decoding together with the LSTM scoring model. We first divide the input tree T into subtrees; each consists of a head h and its dependents. For each subtree, we initialize the agenda with an empty sequence (line 3). A sequence is represented by an LSTM state (line 5), and we also keep track of the total score of the sequence (line 4).

At each step, for each sequence in the agenda, we use a pointer network to calculate the unnormalized attention score between the LSTM state and all the remaining tokens as the scores of attaching each token to the end of the sequence (ATTEND_l in line 14,[1] where $\mathbf{x}_t$ is the vector representation of the token t).

We then create a new sequence for each possible attachment (line 11, where $\oplus$ denotes concatenation), and the score of each new sequence is incremented by the unnormalized attention score (line 12). We also update the corresponding LSTM state of that sequence

[1] All scores are calculated in one go in the actual model, we distribute them in the loop only for readability.

by adding the representation of the attached dependent as input (line 13). The new sequences are then added to the agenda for the next step (line 14).

If the number of new sequences in the new agenda is larger than the beam size, we keep only the highest-scoring ones for further expansion (line 17). When all the sequences are finished, we take the highest-scoring full sequence as the linearization of the subtree (line 19). Finally, after linearizing all subtrees, we combine them into a full sentence as the output (line 21).

Algorithm 1 Left-to-Right linearization.

```
1:  for all h ∈ T do                                      ▷ divide the tree into subtrees
2:      T_h = {h} ∪ dependents(h)                          ▷ all tokens in the subtree
3:      seq = []                                           ▷ initial sequence
4:      seq.score = 0
5:      seq.state = InitialLSTMState()
6:      agenda = [seq]                                     ▷ initial agenda
7:      while |seq| < |T_h| do
8:          for all seq ∈ agenda do
9:              beam = []
10:             for all t ∈ T_h \ seq do                   ▷ a new sequence for each remaining token
11:                 new_seq = seq ⊕ t                      ▷ attach to the right
12:                 new_seq.score += ATTEND(seq.state, x_t)
13:                 new_seq.state = seq.state.addInput(x_t)
14:                 beam.append(seq)
15:             end for
16:         end for
17:         agenda = N-BEST(beam, beam-size)               ▷ keep N-best sequences for next step
18:     end while
19:     S_h = N-BEST(agenda, 1)                            ▷ best sequence for T_h
20: end for
21: S = COMBINESUBTREES(S_1, S_2, ..., S_n)                ▷ combine subtrees into full tree
22: return S
```

Note that the left-to-right decoder can directly order a full tree without the divide-and-conquer strategy. For that, we only need to take lines 2-19 in Algorithm 1, and use T (all tokens in the tree) instead of T_h in line 2. Generally, this would result in slower speed and lower performance since the search space is much larger. We can easily adapt the algorithm to the right-to-left decoding order by changing line 11 to "new_seq = t $\oplus$ seq".

Head-First Generation Order

In most linearization models, the incremental generation algorithm follows the left-to-right sequential order as in Algorithm 1. However, in the linguistic study, the head position often plays a central role in describing the preferences and optimization of word orders (Gibson, 1998; Liu, 2010; Futrell et al., 2015b; Gulordava, 2018). In the linearization models that employ left-to-right generation, such word order properties are only implicitly reflected in the features, if at all. Inspired by the study mentioned above on head-oriented word order preferences, we adopt an improved linearization algorithm. We generate the sequence starting from the head and expanding to both directions. The head-first generation order can easily capture the preferences since it naturally separates the decision into two aspects: (1) which side of the head to append the dependent and (2) which dependent to attach closer to the head, which exactly correspond to the two aspects of the word order preferences, namely (1) the direction of the dependent and (2) the distance of dependent to the head. The algorithm is somewhat similar to He et al. (2009), which also emphasizes the central role of the head by first predicting for each dependent which side of the head it is placed. However, they exhaustively score all permutations, which could be intractable for subtrees with too many dependents, while we use incremental beam-search to guarantee efficiency.

We adapt the left-to-right decoding algorithm to realize the head-first decoding order, as in Algorithm 2. In contrast to the left-to-right decoding, a sequence is now represented by two LSTMs, both initialized with the head representation. The two LSTMs correspond to the left expansion and right expansion of the sequence (line 6-7). At each step, for each sequence in the agenda, we create two expansions to the left and to the right of the current sequence (lines 13 and 17). We use two separate pointer networks to calculate the unnormalized attention scores of the two expansions (lines 14 and 18). We then update the corresponding LSTM state of that sequence by feeding the representation of the attached dependent as input (lines 15 and 19). Although each sequence has two LSTM states, only one state will be updated after each expansion. The rest of the algorithm is identical to the left-to-right decoding.

Training with Beam Search

The head-first linearization introduces spurious ambiguity since there may be two correct attachments (on the left or the right) at each step leading to the same final outcome. Enforcing a canonical sequence of attachments would yield suboptimal performance. We view the order of attaching the dependents (i.e., whether to attach the left or right dependent first) as latent variables while the created sequence is the real target. We adopt the training method in Björkelund and Kuhn (2014): at every step after pruning the agenda, we check if there is still at least one gold partial sequence remaining in the agenda. If not, then we calculate the hinge loss between the highest-scoring gold sequence and *all*

Algorithm 2 Head-first linearization.

```
for all h ∈ T do                                          ▷ divide the tree into subtrees
    T_h = {h} ∪ dependents(h)                              ▷ all tokens in the subtree
    seq = [h]                                              ▷ initial sequence
    seq.score_l = 0
    seq.score_r = 0
    seq.state_l = InitialLeftLSTMState
    seq.state_r = InitialRightLSTMState
    agenda = [seq]                                         ▷ initial agenda
    while |seq| < |T_h| do
        for all seq ∈ agenda do
            beam = []
            for all t ∈ T_h \ seq do        ▷ two new sequences for each remaining token
                seq_l = t ⊕ seq                            ▷ attach left
                seq_l.score_l += ATTEND_l(seq.state_l, v_t)
                seq_l.state_l = seq.state_l.addInput(v_t)
                beam.append(seq_l)
                seq_r = seq ⊕ t                            ▷ attach right
                seq_r.score_r += ATTEND_r(seq.state_r, v_t)
                seq_r.state_r = seq.state_r.addInput(v_t)
                beam.append(seq_r)
            end for
        end for
        agenda = N-BEST(beam, beam-size)
    end while
    S_h = N-BEST(agenda, 1)                                ▷ best sequence for T_h
end for
S = COMBINESUBTREES(S_1, S_2, ..., S_n)      ▷ combine subtrees into full tree
return S
```

incorrect sequences in the agenda.[2]

We also follow the delayed LaSO strategy Björkelund and Kuhn (2014): after all gold partial sequences fall out of the agenda and a loss is incurred, we continue training with the current state by putting the gold partial sequence back into the agenda until finishing generating the full sequence. This is shown to be more sample efficient than the early-update strategy Collins and Roark (2004); Huang et al. (2012), since it allows the model to train on the full sequence, even if the gold path falls out of the agenda early.

The standard hinge loss updates the gold sequence against the incorrect ones by enforcing a margin (typically 1), which punishes all incorrect sequences equally. However, not all incorrect sequences are equally bad in terms of quality; therefore, maintaining a

[2]Perceptron-based training only updates against the highest-scoring incorrect sequences, in contrast, we update against all incorrect sequences, which converged much faster and more stable in the preliminary experiments.

larger margin for worse sequences could improve the performance.

The sequences are typically evaluated with the BLEU score (Papineni et al., 2002), a standard evaluation metric for machine translation. However, we cannot directly use the BLEU score as the margin since it is calculated on the full sentence level while we are training on ordering subtrees, which are fragments of the full sentence. Therefore, we use the inversion number as the proxy, which also punishes the sequence more if it diverges more from the gold sequence.

When we calculate the hinge loss margin for a partial sequence, we first append the remaining unattached tokens to both ends of the sequence in an optimal way according to the gold sequence, then calculate the number of swaps in a bubble-sort to reach the gold sequence (i.e., the inversion number), and take the squared root as the loss.[3] For example, if we have a predicted partial sequence of $(1, 2, 4, 6, 7)$, and the remaining tokens are $\{3, 5\}$, then we first obtain the best available full sequence $(3, 1, 2, 4, 6, 7, 5)$, then calculate the number of swaps in a bubble-sort, which is 4, and the hinge loss margin is thus 2.

Ensemble

The head-first decoding order shows better performance than the left-to-right and right-to-left variants in the preliminary experiments. However, these three different orders have complementary abilities to model the sequence, and each has its strength for certain languages according to their preferences of branching direction. Therefore, we take all three decoding orders to form an ensemble. The three decoders share the same token representation and are trained jointly by summing the losses. During prediction, we select the majority prediction (with random tie-breaking) for each subtree.

3.3.2 Graph-based Decoding

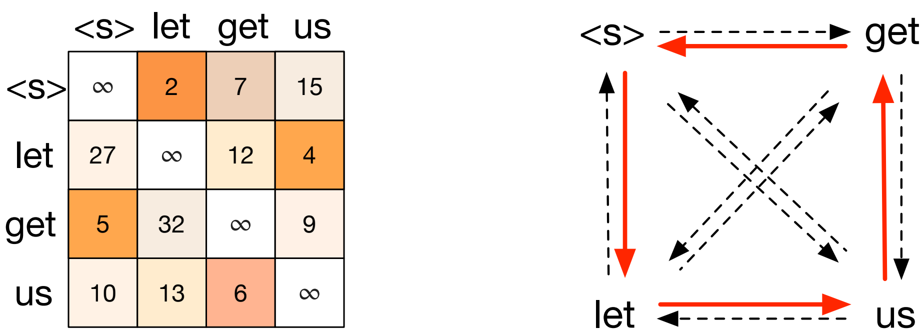
The incremental linearization decoder has a rather slow decoding speed since the neural network model needs to make a calculation for each step of sequence expansion. It also suffers from the error propagation problem as commonly occurred in auto-regressive models. Beam search could alleviate it to some extent, at the cost of more computation and slower decoding. We now consider another approach that performs the calculation only once for the whole tree or for each subtree.

We follow the idea in Knight (1999) to treat linearization as a Traveling Salesman Problem (TSP). The TSP is a task to find a path with the lowest total cost to visit all

[3]Since the expected inversion number is $O(n^2)$, taking the squared root would make it linear and avoid exploding loss values.

the nodes in a graph exactly once and return to the origin. Under the TSP formulation, we need to calculate the cost from every node i to every other node j, which can be interpreted as the log-likelihood of the bigram (i, j). We use the biaffine attention model (Dozat and Manning, 2016) to obtain the costs and use the OR-Tools,[4] an optimization tool for general Vehicle Routing Problems (VRPs), to decode the TSP.

To formulate the linearization task as a TSP, we use a node to represent each input token and an extra node with index 0 as both the origin and destination, which is interpreted as the boundaries in the output sequence. Figure 3.6 demonstrates a decoded TSP graph from its edge cost matrix, where the output sequence is "<s> let us get <s>".



(a) cost matrix **(b)** TSP graph

Figure 3.6: The matrix and graph views of the linearization task as a TSP.

Scoring Model

We use the biaffine attention model (Dozat and Manning, 2016) to calculate the TSP edge costs. The parameters of the decoder consist of two Multi-Layer Perceptrons ($\text{MLP}^{(fr)}$ and $\text{MLP}^{(to)}$) and a biaffine matrix ($\mathbf{W}$). We use the MLPs to obtain different views of the token representation as the first and second token in the bigram:

$$\mathbf{h}_i^{(fr)} = \text{MLP}^{(fr)}(\mathbf{x}_i) \tag{3.7}$$

$$\mathbf{h}_i^{(to)} = \text{MLP}^{(to)}(\mathbf{x}_i) \tag{3.8}$$

We then apply the biaffine transformation on the vectors of the first word $\mathbf{h}_i^{(fr)}$ and the second word $\mathbf{h}_j^{(to)}$ to compute the score $s_{i,j}$ of each bigram (i, j), where $\mathbf{W}$ is the

[4] https://developers.google.com/optimization

weight matrix of size $(k+1) \times (k+1)$, and k is the size of $\mathbf{h}_i^{(fr)}$ and $\mathbf{h}_j^{(to)}$:

$$s_{i,j} = (\mathbf{h}_i^{(fr)} \oplus 1) \times \mathbf{W} \times (\mathbf{h}_j^{(to)} \oplus 1)^\top \tag{3.9}$$

In the actual implementation, we parallelize Equation 3.9, where $\mathbf{S}$ is the output score matrix of size $n \times n$, and n is the number of tokens:

$$\mathbf{S} = (\mathbf{H}^{(fr)} \oplus \mathbf{1}) \times \mathbf{W} \times (\mathbf{H}^{(to)} \oplus \mathbf{1})^\top \tag{3.10}$$

Finally, we turn the score matrix into a non-negative cost matrix for the TSP solver:

$$\mathbf{C} = \max(\mathbf{S}) - \mathbf{S} \tag{3.11}$$

Our model is inspired by the biaffine dependency parser of Dozat and Manning (2016), but stands in contrast in many aspects. They use a bidirectional LSTM to encode the tokens' sequential information, and the biaffine attention itself does not model the sequence. Each cell $s_{i,j}$ in their output matrix $\mathbf{S}$ is interpreted as the score of a dependency arc (i, j). They use a MST algorithm to obtain a tree that maximizes the total score of the arcs in the tree.

In the case of linearization, our input and output are the opposite of theirs. The input has no sequential but syntactic information encoded by the bidirectional Tree-LSTM. Each cell $s_{i,j}$ in the output matrix $\mathbf{S}$ is interpreted as the score of the bigram (i, j). We use a TSP solver to obtain a traversal of the tokens by minimizing the total edge costs, i.e., maximizing the total score of the bigrams in the sequence.

TSP Solver

We use the routing solver of OR-Tools to solve the TSP given the edge costs. It is a generic optimization library for Vehicle Routing Problem (VRP), unlike the more specialized and optimized TSP solvers such as Concorde (Applegate et al., 2006). However, it is more flexible and enables imposing extra word order constraints, e.g., predefined relative order of some tokens.

The solver expects non-negative integers as the cost to minimize, while the scores from the scoring model are real numbers to maximize. Therefore, we need to transform the scores before decoding. Concretely, we turn the scores into non-negative costs by subtracting the maximum value by each score, then multiply the results by a sufficiently large number and round to integer.

We use the GLOBAL_CHEAPEST_ARC strategy to find the initial solution for the TSP. It is a greedy strategy that iteratively selects an edge with the lowest cost that connects two nodes and does not form a cycle, until a full path is found. The solution is then refined with the greedy metaheuristic GREEDY_DESCENT, which attempts to make local changes that reduces the overall cost until reaching a local minimum.[5]

More advanced metaheuristics such as GUIDED_LOCAL_SEARCH (Voudouris and Tsang, 1999) could find better solutions but also require much more decoding time; it is thus less practical for real-time generation tasks. In the preliminary experiments, we find that the greedy strategy can find the optimal solution in the majority of cases with very small time budgets. It works especially well with the aggressive training objective that we will describe next.

Training Objective

We use an aggressive training objective to train the biaffine scoring model; namely, we enforce the score of each bigram (i, j) in the correct sequence z to be higher than any other bigrams in the same row or the same column in the matrix by a margin:

$$\begin{aligned} L = \sum_{(i,j) \in z} \Big(& \sum_{j' \neq j} \max(0, 1 + s_{i,j'} - s_{i,j}) \\ & + \sum_{i' \neq i} \max(0, 1 + s_{i',j} - s_{i,j}) \Big) \end{aligned} \tag{3.12}$$

This objective aims to maximizing the score of each correct bigram (i, j) in both directions, essentially $\log P(j|i)$ and $\log P(i|j)$, where the cells in the same row correspond to all possible tokens following i, and the cells in the column corresponds to all possible tokens preceding j.

The objective is aggressive in the sense that it updates more than "necessary" to decode the correct path. We contrast it to the structured loss in most graph-based dependency parsers (McDonald et al., 2005; Kiperwasser and Goldberg, 2016), which would update the scores of the correct path z against the highest-scoring incorrect path z':

[5]See https://developers.google.com/optimization/routing/routing_options for a comprehensive list of search strategies.

$$L' = \max(0, 1 + \max_{z' \neq z} \sum_{(i',j') \in z'} s_{i',j'} - \sum_{(i,j) \in z} s_{i,j}) \tag{3.13}$$

The non-aggressive objective ensures that the overall score of the correct structure ranks the highest, but not necessarily each single component of the structure has the highest score among the alternatives. It would only work well if there is a exact decoding algorithm that guarantees to find the highest scoring structure, such as the MST algorithm for parsing. However, this is not the case for the TSP.

The aggressive objective for the TSP has two main advantages: (1) it does not require decoding during training, which saves training time; (2) it pushes the scores of each correct bigram to be the highest in the row *and* in the column, which facilitates the greedy solver (`GLOBAL_CHEAPEST_ARC`) to find a good initial solution. In the extreme scenario, if the loss reaches 0, the greedy solver is guaranteed to find the optimal solution since, at each step, the cheapest arc is always a correct bigram instead of any other bigram in the same row or column.

Decoding with Constraints

The order of some types of dependency relations are largely arbitrary (at least syntactically), such as coordination, and the reference in the evaluation is just one of many possible arrangements. If we judge other valid ordering as incorrect, it would not accurately reflect the performance of a model. For this reason, datasets such as SR'19 (Mille et al., 2019) provide the relative word order of some relations to reduce the noise in evaluation. The linearization model should ideally use these information as constraints for decoding.

The constraints does not specify direct adjacency, but general precedence relations. For example, to order the nodes $\{1, 2, 3, 4, 5\}$ with the constraint $2 \prec 3 \prec 1$ and $4 \prec 5$, a valid sequence could be $[2, 4, 3, 5, 1]$, while $[4, 5, 2, 1, 3]$ is invalid.

To incorporate such constraints in the solver, we introduce an additional variable associated with each node in the TSP routing problem, where the value is incremented by one after each step. In other words, if a node is visited in the n-th step, then the associated variable will have the value n. Then we add inequality constraints about those variables specified in the word order information into the routing problem and let the solver find the path that satisfies the constraints.

In practice, the solver can always find a solution to linearize the subtrees with the constraints since the problem size is quite small. However, it sometimes cannot find any solution to directly linearize the full tree within the time limit (1-10% of the cases depending on the treebank) because there are more nodes and more constraints in the full tree. In this case, we simply remove the constraints and rerun the solver.

3.3.3 Transition-based Decoding

Transition System

Finally, we introduce a third decoding algorithm, which takes an initial sequence as input, and reorders the tokens in the sequence step by step until reaching the final order. The reordering decoder works similarly to the bubble-sort algorithm, which iteratively compares two neighboring tokens and decides whether to swap them. We define a transition system for the reordering with two transitions, *swap* and *shift*, which is inspired by Nivre (2009) for non-projective dependency parsing, as shown in Table 3.2.

Transition	Before		After	Precondition
shift	$\langle\sigma, \quad t_i \mid \beta\rangle$	$\Rightarrow$	$\langle\sigma \mid t_i, \quad \beta\rangle$	
swap	$\langle\sigma \mid t_i,\ t_j \mid \beta\rangle$	$\Rightarrow$	$\langle\sigma, \quad t_j \mid t_i \mid \beta\rangle$	ROOT $\prec t_i \prec t_j$

Table 3.2: The SWAP transition system.

In the transition system, a configuration consists of a stack σ, which is initially empty, and a buffer β, which initially holds all input tokens. The *shift* transition moves the front of the buffer to the top of the stack, and the *swap* transition moves the top of the stack back to the second place in the buffer. When all tokens are moved from the buffer to the stack, the procedure terminates. To prevent the model predicting infinite *shift-swap* loops, we only allow *swap* if the initial index of the top of the stack is smaller than the front of the buffer.

Scoring Model

We then implement a model to predict the transitions given the configurations. We use two LSTMs to dynamically encode the stack from left to right (LSTM_σ) and the buffer from right to left (LSTM_β), which is inspired by Dyer et al. (2015). We then concatenate the two outputs and use an MLP to predict the next transition.

When a *shift* is performed, we update the LSTM_σ state with the vector of the shifted token as the new stack representation, and the new buffer representation is the LSTM_β output of the new front token in the buffer; when a *swap* is performed, the new stack representation is the LSTM$_\sigma$ output of the new top token in the stack, and the new buffer representation is recalculated by feeding the now second and first token in the buffer to the LSTM$_\beta$ state of the third token.

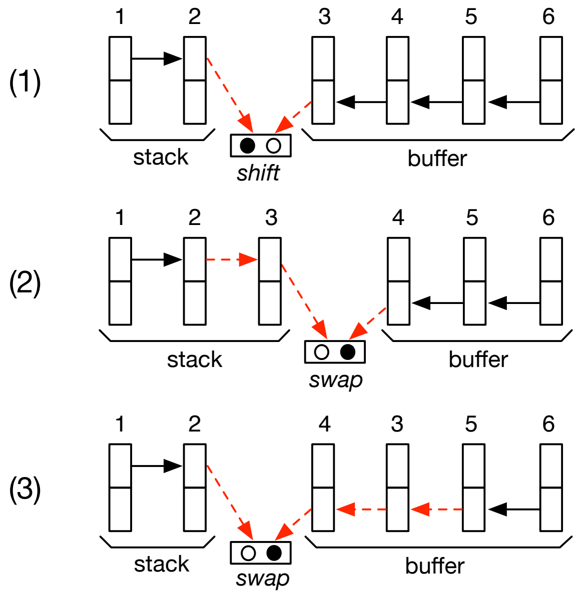


Figure 3.7: An illustration of the swap model. (1) shows a configuration [1 2 | 3 4 5 6], where the stack contains 1 and 2, the buffer contains 3, 4, 5, and 6, and the model predicts *shift* to move 3 from the buffer to the stack; (2) is the resulted configuration, and the model predicts *swap*, which moves 3 back to the buffer behind 4, as shown in (3). The solid black arrows represent the computation already done in the previous steps, and the dashed red arrows represent the new computation needed for the current step.

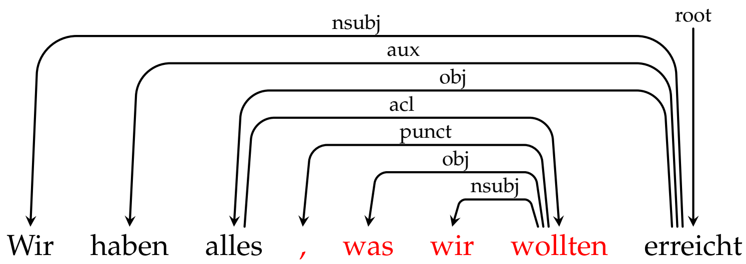
Figure 3.7 illustrates the model under the transition system, where the arrows to the right represent LSTM$_\sigma$, the arrows to the left represent LSTM$_\beta$, and the arrows between the stack and the buffer represent the MLP. After each transition, little computation is needed to represent the new stack and buffer, marked with the red dashed line. The example illustrates the steps to modify the configuration [1 2 | 3 4 5 6] into [1 2 | 4 3 5 6].

Producing Non-Projective Sentences

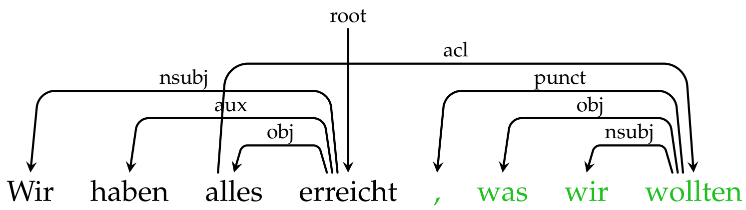
The transition system is sound and complete, which means there is always a sequence of transitions to sort any sequence into any target reordering. However, using the transition system directly for the full linearization task yields suboptimal performance in practice. Due to the noisy input order, it is difficult for the LSTM model to learn good representations for the random stack and buffer and predict correct transitions (cf. Vinyals et al. (2015a) for the discussion on encoding a set with an LSTM).

However, the transition-based decoding is still useful in another scenario, namely we only use it as post-processing to reorder a linearized projective sentence into a non-projective one. In this case, the input sequence is meaningful and consistent (assuming they are in near-correct projective order), it is much easier for the model to learn the representation of the input and the decoding transitions. Also, the number of required transition steps of post-processing is much lower than that of a full ordering. The average-case complexity for the full ordering is $\mathcal{O}(n^2)$, where n is the number of the tokens, while the complexity of the post-processing is near-linear.

Figure 3.8 illustrates how the transition system works, in which (a) is a tree in its correct projective order and (b) is the non-projective sentence after reordering. The subtree *was wir wollten* (what we wanted) is a relative clause that modifies *alles* (all), and should be extraposed after the main clause *Wir haben alles erreicht* (we have achieved everything). Table 3.3 illustrates the transitions that are needed to reorder the projective sentence.



(a) a projective sentence



(b) the non-projective sentence after reordering

Figure 3.8: An pair of projective and non-projective sentences.

Using the transition system as a post-processing step stands in contrast to Bohnet et al. (2012), where they pre-process the tree by lifting the arcs so that the correct word order could form a projective tree. These two approaches draw inspiration from the non-projective parsing in Nivre (2009) and the pseudo-projective parsing in Nivre and Nilsson (2005) respectively. We argue that our post-processing approach is more convenient since there is no need to change the original tree's syntactic annotation. It is also much easier to evaluate the effectiveness of the sorting model.

Transition	Stack	Buffer
	[]	[Wir haben alles , was wir wollten erreicht]
shift	[Wir]	[haben alles , was wir wollten erreicht]
shift	[Wir haben]	[alles , was wir wollten erreicht]
shift	[Wir haben alles]	[, was wir wollten erreicht]
shift	[Wir haben alles ,]	[was wir wollten erreicht]
shift	[Wir haben alles , was]	[wir wollten erreicht]
shift	[Wir haben alles , was wir]	[wollten erreicht]
shift	[Wir haben alles , was wir wollten]	[erreicht]
swap	[Wir haben alles , was wir]	[erreicht wollten]
swap	[Wir haben alles , was]	[erreicht wir wollten]
swap	[Wir haben alles ,]	[erreicht was wir wollten]
swap	[Wir haben alles]	[, erreicht was wir wollten]
shift	[Wir haben alles ,]	[erreicht was wir wollten]
shift	[Wir haben alles , erreicht]	[was wir wollten]
shift	[Wir haben alles , erreicht was]	[wir wollten]
shift	[Wir haben alles , erreicht was wir]	[wollten]
shift	[Wir haben alles , erreicht was wir wollten]	[]

Table 3.3: The transitions to reorder the projective tree in Figure 3.8.

3.4 Summary

We have introduced the two major components for the linearization system: (1) an encoding representation and (2) a decoding algorithm coupled with a scoring model.

We develop four different encoders that treat the input as four different structures: bag-of-words, sequences, tree, and graph. The bag-of-words encoder is the simplest one, but it completely ignores the syntax. The sequence encoder needs to first convert the tree into a sequence of tokens then encode it, which requires the model to learn all the different random sequences representing the same tree. The tree encoder exactly fits the input structure, which is the model that we propose to use. Finally, the graph encoder treats the input as a graph and models it in a too generalized way than needed.

To decode the output sequence, we propose three different algorithms: incremental, graph-based, and transition-based. The incremental decoding algorithm uses LSTMs to represent and score the partial sequences, which requires as many steps of computation as the number of tokens in the tree. The graph-based decoding algorithm treats the linearization task as a TSP and calculates the scores of all possible bigrams with a biaffine attention model. Although the number of bigrams is quadratic to the number of tokens, it could be parallelized and calculated all at once, therefore more efficient in practice. The transition-based decoding algorithm iteratively swaps the token in the initial sequence until reaching the final predicted sequence. It has a theoretically quadratic complexity if the initial sequence is random. However, if we only use it as a post-processing method to produce non-projective sentences, it runs in linear time empirically.

As we will show in Chapter 4, the optimal linearization model consists of the Tree-LSTM encoder, the graph-based TSP decoder, and additionally the transition-based decoder as post-processing. The workflow is illustrated in Figure 3.9.

It takes an unordered dependency tree as input and produces a linearized sentence as output, which is potentially non-projective, i.e., with crossing dependency arcs. In the pipeline, we (1) divide the tree into subtrees, (2) linearize each subtree by solving a TSP, (3) combine the linearized subtrees into a projective tree, and (4) use the transition system to obtain a non-projective tree.

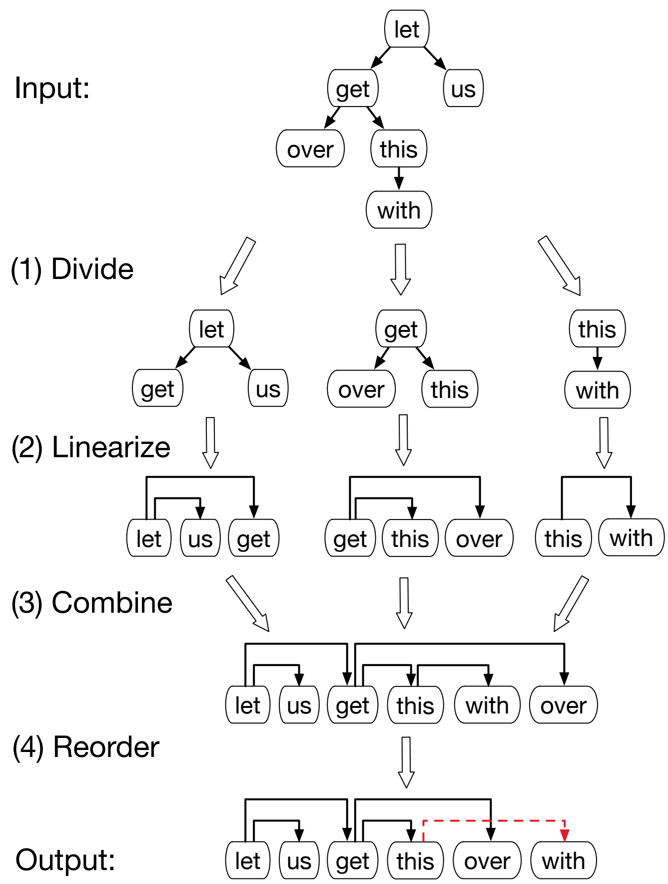


Figure 3.9: An overview of the optimal linearization system.

Chapter 4

Linearization Experiments

In this chapter, we conduct various experiments for several purposes. First, we need to determine the optimal configuration of the linearization model discussed in the previous chapter, including (1) which encoder to represent the input dependency tree, (2) which decoder to produce the linear order, (3) decoding on the subtree or tree level, (4) how to generate non-projective tree.

We then analyze the robustness of the decoding algorithm by identifying the extent of model errors and search errors, and control the training data size for a more meaningful cross-lingual comparison, which also demonstrates our model's data efficiency.

Finally, we look into the cases where its prediction deviates from the reference to understand the difficulty of linearization categorized by the dependency relations and the language families. To distinguish word order variations from genuine mistakes, we manually inspect the predictions in a few languages to understand to what extent the deviations are a result of word order variations, which cannot be reflected in the automatic evaluation metrics.

4.1 Data and Evaluation Metrics

We conduct the experiments on a subset of treebanks from the Universal Dependencies v2.4 (Nivre et al., 2019). Specifically, we select the treebanks with at least 500 sentences in the training set for more stable statistics. If there are multiple treebanks for one language, we only select the largest one that is not delexicalized.[6] In total, there are 53 languages.

Table 4.1 gives an overview of the treebanks used in the experiments. Despite UD's effort toward language diversity, the majority of the languages in the data sets belong

[6]Several large treebanks in the UD are delexicalized, as the text is from a copyrighted corpus.

treebank	language	(sub-)family	sents	len-sent	len-sub	np-sent	np-arc
af_afribooms	Afrikaans	IE, Germanic	1,315	25.77	3.41	22.21	2.17
ar_padt	Arabic	Semitic	6,075	36.85	3.00	8.13	0.47
bg_btb	Bulgarian	IE, Slavic	8,907	13.96	3.20	3.17	0.59
ca_ancora	Catalan	IE, Romance	13,123	31.82	3.51	4.43	0.33
cs_pdt	Czech	IE, Slavic	68,495	17.13	3.21	11.38	1.94
cu_proiel	Old Slavonic	IE, Slavic	4,124	9.08	2.97	16.39	3.78
da_ddt	Danish	IE, Germanic	4,383	18.34	3.50	21.83	3.01
de_gsd	German	IE, Germanic	13,814	19.10	3.46	9.52	0.96
el_gdt	Greek	IE, Greek	1,662	25.47	3.47	5.96	0.50
en_ewt	English	IE, Germanic	12,543	16.31	3.48	5.28	0.61
es_ancora	Spanish	IE, Romance	14,305	31.08	3.50	5.28	0.42
et_edt	Estonian	Uralic, Finnic	24,384	13.99	3.24	3.17	0.57
eu_bdt	Basque	Basque	5,396	13.52	3.15	33.52	7.21
fa_seraji	Persian	IE, Iranian	4,798	25.23	3.21	6.86	0.69
fi_ftb	Finnish	Uralic, Finnic	14,981	8.52	2.92	7.69	2.04
fr_gsd	French	IE, Romance	14,450	24.54	3.52	5.09	0.41
fro_srcmf	Old French	IE, Romance	13,909	9.78	3.08	15.41	3.57
gl_ctg	Galician	IE, Romance	2,272	34.92	3.57	0.00	0.00
got_proiel	Gothic	IE, Germanic	3,387	10.34	2.97	16.77	3.64
grc_proiel	Ancient Greek	IE, Greek	15,014	12.46	3.05	37.39	7.70
he_htb	Hebrew	Semitic	5,241	26.28	3.21	7.63	0.52
hi_hdtb	Hindi	IE, Indic	13,304	21.13	3.40	13.81	1.20
hr_set	Croatian	IE, Slavic	6,915	22.11	3.29	8.88	0.83
hu_szeged	Hungarian	Uralic	910	22.16	3.33	25.71	3.27
hy_armtdp	Armenian	IE, Armenian	672	23.15	3.28	9.08	0.87
id_gsd	Indonesian	Austronesian	4,477	21.78	3.23	5.03	0.57
it_isdt	Italian	IE, Romance	13,121	21.04	3.48	1.31	0.10
ja_gsd	Japanese	Japanese	7,125	22.49	3.54	0.00	0.00
ko_kaist	Korean	Korean	23,010	12.88	2.71	21.92	3.59
la_ittb	Latin	IE, Latin	16,809	17.45	3.34	37.78	5.34
lt_alksnis	Lithuanian	IE, Baltic	1,056	16.58	3.17	13.64	2.39
lv_lvtb	Latvian	IE, Baltic	9,664	16.39	3.20	6.37	1.08
lzh_kyoto	Classical Chinese	Sino-Tibetan	7,056	4.87	2.56	0.00	0.00
mt_mudt	Maltese	Semitic	1,123	20.37	3.30	2.85	0.30
nl_alpino	Dutch	IE, Germanic	12,269	15.16	3.46	14.43	2.64
no_bokmaal	Norwegian	IE, Germanic	15,696	15.54	3.43	7.52	0.97
orv_torot	Old_Russian	IE, Slavic	13,336	8.90	2.95	15.12	3.80
pl_pdb	Polish	IE, Slavic	17,766	16.00	3.14	6.50	0.95
pt_gsd	Portuguese	IE, Romance	9,664	26.46	3.47	5.69	0.44
ro_nonstandard	Romanian	IE, Romance	10,144	19.93	3.48	5.56	0.54
ru_syntagrus	Russian	IE, Slavic	48,814	17.83	3.16	7.53	1.09
sk_snk	Slovak	IE, Slavic	8,483	9.50	3.17	3.15	1.05
sl_ssj	Slovenian	IE, Slavic	6,478	17.37	3.43	13.49	2.48
sr_set	Serbian	IE, Slavic	3,328	22.31	3.31	4.72	0.40
sv_talbanken	Swedish	IE, Germanic	4,303	15.49	3.34	3.09	0.37
te_mtg	Telugu	Dravidian	1,051	4.84	2.83	0.10	0.04
tr_imst	Turkish	Turkic	3,664	10.31	2.93	11.05	3.00
ug_udt	Uyghur	Turkic	1,656	11.63	2.94	4.41	0.73
uk_iu	Ukrainian	IE, Slavic	5,496	16.81	3.23	7.99	1.27
ur_udtb	Urdu	IE, Indic	4,043	26.88	3.51	23.00	1.72
vi_vtb	Vietnamese	Austro-Asiatic	1,400	14.49	3.22	3.14	0.51
wo_wtb	Wolof	Niger-Congo	1,188	19.83	3.35	2.69	0.22
zh_gsd	Chinese	Sino-Tibetan	3,997	24.67	3.33	0.63	0.06
MIN			672	4.84	2.56	0.00	0.00
MAX			68,495	36.85	3.57	37.78	7.70

Table 4.1: Information about the treebanks used in the experiments, including the language (sub-)family, data size, average sentence length, average non-leaf subtree size, percentage of non-projective sentences and non-projective arcs.

to the Indo-European (IE) family with 36 out of 53 languages, dominated by the three major subfamilies: Slavic (11), Germanic (8), and Romance(8). The training set size ranges from 672 (Armenian) to over 68,495 (Czech) sentences, which differs in two orders of magnitude. The average sentence length also varies a lot, from 5 words (Classical Chinese and Telugu) to 37 words (Arabic), which has the greatest impact on the evaluation results, since longer sentences are more difficult to both encode and decode. In contrast, the average (non-leaf) subtree size is very consistent across all languages, ranging from 2.6 to 3.6. In terms of projectivity, some treebanks are completely projective (Galician, Japanese, and Classical Chinese), some have over one-third of non-projective sentences (Basque, Ancient Greek, and Latin).

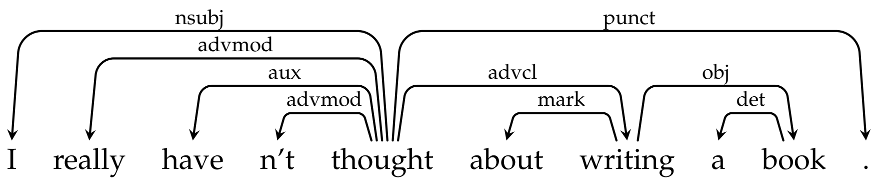
As evaluation metrics, we use the BLEU score (Papineni et al., 2002), which calculates the precision of n-grams between the generated and reference sentences. It is a commonly used metric in machine translation since it gives a relatively comprehensive similarity estimation and correlates relatively well with human judgements (Coughlin, 2003).

We also include the exact matching rate on the full sentence level (EMF) and the subtree level (EMS). EMF is straightforward to understand but could be too harsh against minor variations. EMS evaluates the accuracy of each subtree, and we only consider the non-trivial cases where the head has at least one dependent (non-leaf subtrees). It can perfectly reflect the quality of the subtree-level linearizers. Also, unlike EMF, it is less sensitive to the length of the sentence, since the size of each subtree has much smaller variation within a treebank and among treebanks. The only drawback of EMS is that it does not capture errors regarding projectivity.

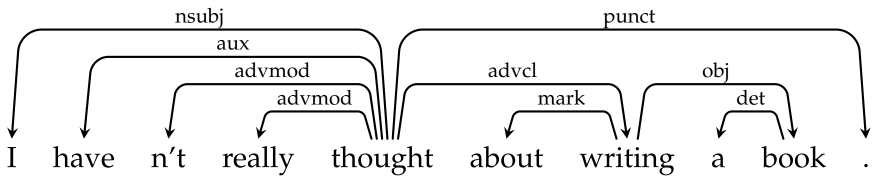
Figure 4.1 gives a pair of reference and predicted sentences as an example. The BLEU score compares the n-gram precision with n up to 4, which are $10/10 = 1$ for unigrams, $6/9 = 0.6667$ for bigrams, $4/8 = 0.5$ for trigrams, and $3/7 = 0.4286$ for quadrigrams, and the final BLEU score is their geometric mean $\sqrt[4]{1 \times 0.6667 \times 0.5 \times 0.4286} = 0.6147$. The EMF simply checks if the prediction exactly matches the reference, which is 0. The EMS calculates the proportion of matched (non-leaf) subtrees, which is $2/3 = 0.6667$.[7]

Note that we compare the linearization models' output with a single reference, which could penalize valid alternative word order, e.g., in the cases of coordination or topicalization. In fact, The prediction in the previous example is a grammatical alternative order although the meaning differs slightly from the reference. Therefore, we also conduct manual inspection on a subset of languages to shed some light on the extent of the word order variation's influence on the evaluation.

[7]By convention, these numbers are typically presented as percentages, ranging from 0 to 100.



(a) reference



(b) prediction

Figure 4.1: An example to demonstrate the different evaluation metrics, where (a) is the reference sentence and (b) is the predicted sentence.

4.2 Encoders

The first question while designing the linearization model is how to represent the input dependency tree. We experiment with several encoders that model the input as different types of structures: bag-of-words, sequence, tree, and graph. The bag-of-words encoder (`bag`) only relies on the token-level representation and each token is not aware of other tokens in the tree. The sequence encoder (`seq`) treats the random serialization of an unordered dependency tree as a sequence and applies a bidirectional LSTM to obtain the representation. The tree encoder (`tree`) directly models the input dependency tree by using a bidirectional Tree-LSTM to propagate the information from the dependents to the heads and then from the heads to the dependents. The graph encoder (`graph`) treats the input as a general graph. It uses a GAT to propagate the information to each node from its neighbors (heads and dependents), which in turn receive information from their neighbors further away.

We experiment with the combination of each encoder and the TSP decoder under two conditions: with or without the divide-and-conquer strategy (denoted as `TSP-sub` and `TSP-full`). The average results are shown in Table 4.2, and the detailed results are listed in Tables A.1 to A.3 in the Appendix A. With the subtree TSP decoder, the tree encoder achieves the best performance, 0.6 points higher than the sequence encoder, and on par with the graph encoder. The performance gap among the encoders is much larger with the full TSP decoders, i.e., without the divide-and-conquer strategy. The disadvantage of the bag-of-words encoder is especially apparent in this condition, and the performance

Decoder	Encoder	BLEU	EMF	EMS
TSP-sub	bag	76.20	36.62	84.21
	seq	77.41	38.13	85.03
	tree	78.04	39.61	85.63
	graph	78.04	39.44	85.56
TSP-full	bag	34.80	16.81	49.49
	seq	73.05	36.04	81.28
	tree	73.60	36.30	81.33
	graph	68.98	33.02	77.53

Table 4.2: Different encoders combined the subtree and full tree TSP decoders, evaluated on the BLEU score and the exact matching rates of subtrees and full trees.

drop of the graph decoder is also quite significant. We believe the reason for the graph decoder' large performance drop is that it can only model the context of a node a limited number of steps away due to the model's design. In contrast, both LSTM and Tree-LSTM can theoretically model contexts from unlimited distances/depth. This is not a serious problem on the subtree-level decoding since each subtree has only a few dependents, and the local context could suffice to determine there relative positions. However, when decoding on the full tree level, the deeper contexts become more important.

The full TSP decoder orders all words of the full tree at once, which is a much larger search problem and much more difficult to model. This explains the large performance drop of the bag-of-words and the graph decoder; it also clearly demonstrates the tree encoder's superior representation ability. Interestingly, in both conditions, the performance of the sequence decoder follows the tree encoder quite closely. This suggests that the sequential LSTM can approximate hierarchical information to some extent.

In the sequential LSTM encoder, we introduced the scoping brackets so that the tree structure can be serialized unambiguously (see §3.1.2). We now show how important it is to the encoder in an ablation experiment. We experiment with different types of input to the sequence encoder. The first one (`seq-nb`) is the sequence without scoping brackets, which is the pre-order traversal of the dependency tree with the dependents in random order. The second one (`seq-rd`) does not retain any tree information at all, which is just a random reordering of all the tokens in the tree. We compare the performance of these modifications to the proper sequence encoder (`seq`) and the bag-of-words encoder (`bag`) where the sequence is not considered at all. The results are as shown in Table 4.3.

Again, the performance difference is clearer with the full tree TSP decoder. The sequential encoder's performance drops significantly without the scoping bracket (by 21

Decoder	Encoder	BLEU	EMF	EMS
TSP-sub	seq	77.41	38.13	85.03
	seq-nb	75.98	36.03	83.86
	seq-rd	75.73	35.32	83.63
	bag	76.20	36.62	84.21
TSP-full	seq	73.05	36.04	81.28
	seq-nb	52.16	23.57	65.65
	seq-rd	33.24	16.01	48.44
	(bag)	34.80	16.81	49.49

Table 4.3: Different versions of the sequence encoders combined the subtree and full tree TSP decoders, evaluated on the BLEU score and the exact matching rates of subtrees and full trees.

BLEU points); it further deteriorates by 19 BLEU points when the full random sequence is used as the input. The exact matching metrics show the same trend. These results demonstrate the importance of an unambiguous representation of the input tree traversal. Interestingly, using the sequential encoder to encode the fully random sequence performs even worse than not encoding the sequence at all (as in the bag-of-words encoder) since the model is forced to learn noises.

4.3 Decoders

In §4.2 we have established that the Tree-LSTM encoder has the best performance among the alternatives. We now experiment with different decoders using the same Tree-LSTM encoder. Again, for each decoder, we consider the variants on the subtree and tree level.

We start with a random linearization baseline (Rand). In the random subtree linearization, we randomly order each subtree and ensure the output is projective. In contrast, in the random full tree linearization, the order is completely random and most likely non-projective. We then test the incremental decoders with different generation orders: left-to-right (L2R), right-to-left (R2L), and head-first (H2D). Since the head-first linearization involves the head of each subtree, it does not apply to ordering the full tree directly. We include the TSP decoder in both subtree and tree levels. Finally, we include the transition-based decoder on the full tree level (SWAP), which directly reorders a random input order toward the correct order.

The average results are shown in Table 4.4, and the detailed results are listed in Tables A.4 to A.6 in the Appendix A. The difference between random baselines on the subtree level and the tree level (11.86 vs 2.96) clearly shows the benefit of the projec-

level	decoder	BLEU	EMF	EMS
sub	`Rand`	11.86	1.95	23.59
	`L2R`	76.89	38.21	84.79
	`R2L`	76.71	37.02	84.24
	`H2D`	77.14	38.00	84.92
	`TSP`	78.04	39.61	85.63
full	`Rand`	2.96	1.83	23.47
	`L2R`	71.29	35.56	80.07
	`R2L`	71.93	35.61	80.40
	`TSP`	73.60	36.30	81.33
	`Swap`	55.63	25.66	71.34

Table 4.4: Performances of different decoders on the subtree and full tree level.

tive restriction. The two random baselines do not involve any modeling and searching; they simply reflect the distribution of projective and non-projective sentences in the treebanks. Since the majority of sentences are projective, a random baseline that only produces projective sentences are more likely to be correct than a random baseline that also produces non-projective sentences. This explains one advantage of the subtree-level decoders, namely, the inductive bias toward projectivity benefits from the data distribution.

All the decoders on the subtree level outperform their counterparts on the tree level. Among the subtree decoders, the TSP decoder clearly outperforms the variants of incremental beam search decoder in all metrics. The head-first decoder has a small advantage over the left-to-right and right-to-left decoders. Interestingly, the TSP decoder performs on par with the incremental decoders on the full tree level. This suggests that, when the search space becomes larger, the simple bigram scoring model with TSP-based decoding starts losing its advantage over the more sophisticated LSTM scoring model with incremental decoding.

Finally, the `Swap` decoder can correct the random input order to some extent but performs significantly lower than the incremental and graph-based decoders. We believe the main reason is that the average number of steps needed to order a random sequence is quadratic to the length of the sequence, which could be hundreds or even thousands of steps for a long sentence. This would incur serious error propagation problems, i.e., one mistake earlier in the sequence would increase the chance of making further mistakes. Therefore, we do not use the `Swap` decoder to order the input directly. Instead, we only use it to reorder a projective sentence, which would require much shorter transition sequences, thus causing much smaller error propagation.

4.4 Model Errors vs. Search Errors

As previously shown, the subtree-level decoders consistently outperform their tree-level counterparts. The reason is attributed to the reduced search space of the divide-and-conquer strategy. We now take a closer look at the errors by the two TSP decoders to better explain the benefit of this strategy in terms of error reduction.

There are two types of errors in search problems with machine learning models: **model errors** and **search errors**. Model errors are due to the model's lack of generalization ability resulting in the correct solution having a lower score than the incorrect solution; search errors occur when the search algorithm is not optimal and does not find the correct solution even though it has a higher score. The TSP decoders use a machine learning model and a greedy solver, which could incur both types of errors. Therefore, we quantify how often each type of error occurs in the `TSP-sub` and `TSP-full` decoders. Note that the model error rate is normalized by the total number of instances, while the search error rate is normalized by the number of instances where the model is correct. For `TSP-sub`, we consider the errors on both subtree and tree level, where a tree-level error is counted if any of its subtrees has an error. In this way, we can directly compare the two TSP decoders. The results are shown in Table 4.5.

Decoder	level	model error	search error
`TSP-sub`	sub	12.51%	0.07%
`TSP-sub`	full	58.25%	1.32%
`TSP-full`	full	57.87%	10.27%

Table 4.5: Model and search error rates by the `TSP-sub` and `TSP-full` decoders.

Generally, for both models, the magnitude of search errors is much smaller than the model errors, especially for `TSP-sub` (with a negligible 0.07%). This indicates that the greedy TSP solver is fully competent for this task. On the tree level, the model errors of both decoders are comparable, which means that the scoring model can learn the word order equally well regardless of the size of the problem. However, `TSP-sub` has significantly fewer search errors than `TSP-full`, which is the second reason for the performance difference between these two decoders.[8] In this respect, a more optimized TSP solver could improve `TSP-full` at the cost of slower decoding speed, while the greedy solver is quite sufficient for `TSP-sub`.

[8] The first reason is the inductive bias of projectivity as shown in the comparison between two random baselines.

4.5 Non-Projectiviy

The subtree-level decoders generally perform better than the tree-level decoders, but using them restricts the output to be projective sentences. To remedy this restriction, we introduce a transition-based reordering model to adjust the projective sentences into potentially non-projective ones.

Table 4.6 shows the comparison between the TSP decoders on the full tree and subtree level and the additional reordering for the subtree decoder. As already shown before, `TSP-sub` outperforms `TSP-full` by a large margin, and adding the reordering leads to a further improvement of 0.5 BLEU score. More importantly, it improves 1% on the exact match since many predicted sentences that should be non-projective already have the correct projective order. The exact matching rate on the subtree has almost no difference since the reordering mainly only moves the tokens to create non-projective arcs but generally does not affect the subtrees' local order.

Decoder	BLEU	EMF	EMS
`TSP-full`	73.60	36.30	81.33
`TSP-sub`	78.04	39.61	85.63
`TSP-swap`	78.55	40.60	85.61

Table 4.6: TSP decoders on the full tree and subtree level and additionally with reordering, evaluated on the BLEU score and the exact matching rates of subtrees and full trees.

To further demonstrate the effect of reordering on the non-projective trees, figure 4.2 shows the improvement in EMF after the reordering with respect to the percentage of non-projective sentences in each treebank. Generally, the more non-projective sentences a treebank has, the more improvement it benefits from the reordering. Also, there is almost no negative impact from the reordering, even for treebanks without non-projective sentences, because the reordering model simply learns to always predict shift.

Table 4.7 summarizes the change of EMF after reordering the output of the `TSP-sub` decoder. Among the predicted sentences in all treebanks, 1.86% of them have the correct projective order (i.e., each subtree's order is correct) but not exactly correct. 1.17% of all sentences are non-projective and correct after reordering, which corresponds to a 60.30% success rate of the reordering model. Among all sentences, 1.20% are turned from incorrect to incorrect,[9] while 0.22% of them are originally projective and correct but turned

[9] A tiny fraction (0.03%) of sentences are corrected by the reordering, which is not due to the added non-projectivity but corrected subtree orders. However, we believe this is just by accident since the reordering model is never trained to correct the errors in the subtree order.

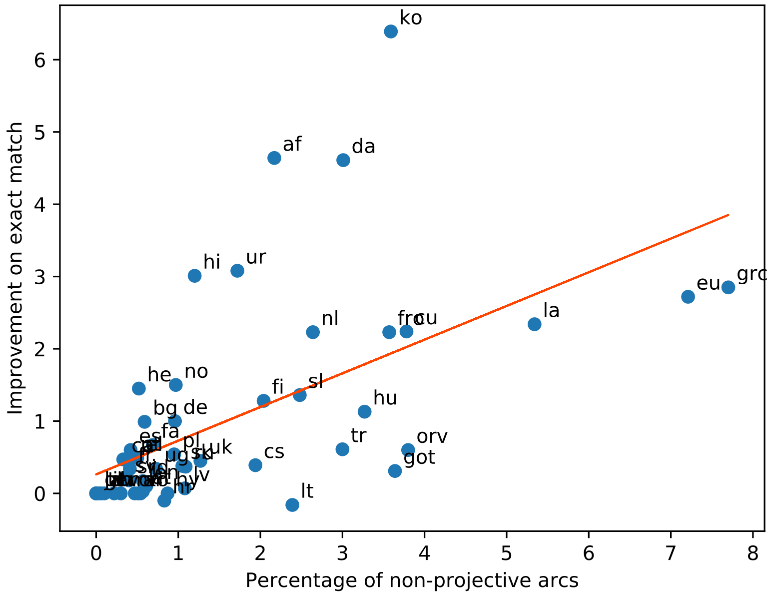


Figure 4.2: The improvement of full tree match after reordering with respect to the percentage of non-projective arcs in each treebank.

incorrect by the reordering. This shows that the reordering model does not always make things better; however, the benefit clearly prevails.

correct projective order	1.86%
correct non-projective	1.17%
incorrect to correct	1.20%
correct to incorrect	0.22%

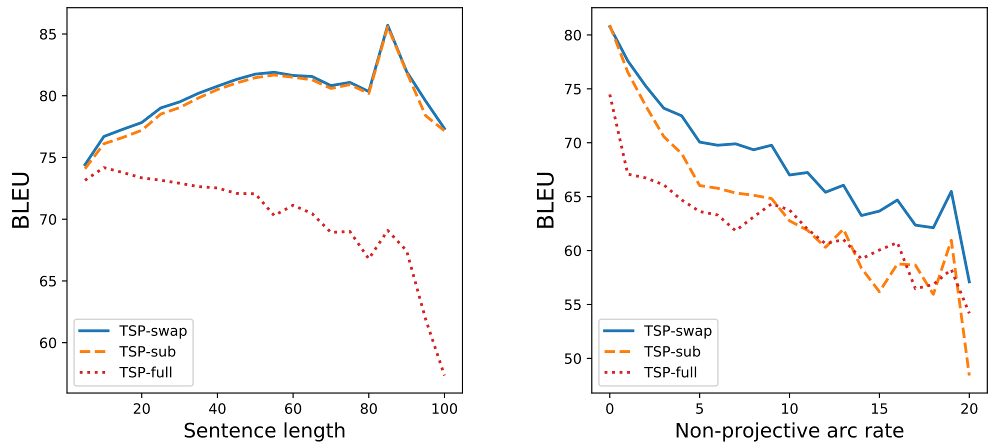
Table 4.7: Percentage of sentences that are changed better or worse by the reordering.

Finally, we conduct a detailed comparison of different TSP decoders on sentences with different lengths and percentages of non-projective arcs. Figure 4.3a shows the BLEU score with respect to the sentence length. The `TSP-sub` model performs consistently with different sentence lengths, while the `TSP-full` model performs much worse on longer sentences. This again shows that the subtree decoder's divide-and-conquer strategy can reduce search errors for large TSP problems. Finally, post-processing with the swap system (`TSP-swap`) consistently improvements `tsp` across all sentence lengths.

Figure 4.3b shows the BLEU score with respect to the percentage of non-projective arcs in the gold tree. Clearly, `TSP-sub` performs lower than `TSP-full` for sentences with more non-projective arcs due to the projectivity restriction. In contrast, the overall

BLEU score of `TSP-sub` is higher since 99% of the arcs and 90% of the sentences are projective. With the help of the swap system, `TSP-swap` completely surpasses `TSP-full` on the non-projective sentences.

In sum, the `TSP-swap` model shows clear advantages over the other models since it is less prone to search error due to the reduced TSP size and free from the projectivity restriction, it is thus the best of both worlds.



(a) BLEU vs. sentence length.

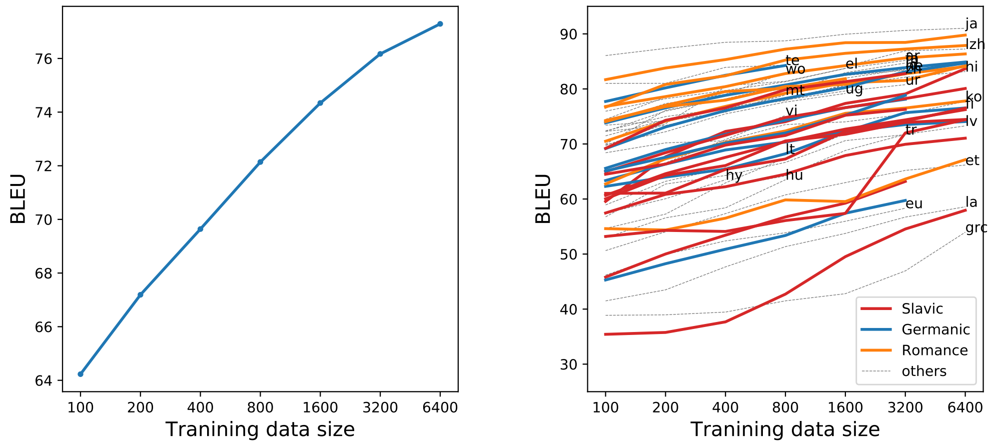
(b) BLEU vs. non-projective arc rate.

Figure 4.3: BLEU score with respect to the sentence length and the non-projective arc rate.

4.6 Training Data Sizes

In our datasets, the treebank sizes range quite a lot, which contributes to the factors (along with sentence length, genre, etc.) that obstruct a meaningful comparison of linearization performance among the languages. Unlike the other factors, this is a variable that we could easily control.

We train a `TSP-sub` model on the first n sentences in the training set for each language, where n ranges from 100, 200, 400, up to 6400. We then plot the change of BLEU scores with respect to the training data size in Figure 4.4, where (a) shows the average trend and (b) shows each individual language. The detailed results are listed in Tables A.4 to A.6 in the Appendix A. For ease of visualization, we color the three major Indo-European subfamilies (Slavic, Germanic, and Romance) and note the language codes of the rest.



(a) average **(b)** individual treebanks

Figure 4.4: BLEU scores of models trained on different data sizes (on a logarithmic scale), where (a) shows the average and (b) shows individual treebanks.

On average, the model only has a moderate performance drop even with a minimal training data size. On the one hand, this demonstrates our model's ability to generalize with very few training instances; on the other hand, it also indicates that there is great regularity between the structural order and linear order since.

After controlling for the data size factor, there is still a considerable variation of the BLEU scores among languages, especially in the low-resource scenario (from 35 for Old Russian to 85 for Japanese). Among the three subfamilies, Slavic languages tend to have lower BLEU scores (35 to 70), Germanic languages lie in the middle range (45 to 75), and Romance languages generally have higher scores (55 to 80).

4.7 Error Analysis

Previously, we have analyzed the prediction errors from the perspective of modeling and searching, we now analyze the error cases in terms of dependency relations to better understand the difficulties in linearization from the linguistic perspective. From the We can interpret the error rate as a proxy of word order freedom since we have a highly accurate model, which means the predictions deviating from the reference have good chances of being correct alternative order.[10]

[10]We can directly measure the word order freedom in a delexicalized and context-free manner, i.e., by calculating the entropy of each dependency relation as in Yu et al. (2019b). However, it is a highly simplified

We separate the errors into two main types: head error and sibling error. A head error is when the dependent is on the wrong side of its head. A sibling error is when the dependent is on the correct side, but its direct neighbors do not have the correct local order, i.e., the preceding word should follow it or the following word should precede it. Figure 4.5 illustrates the two types of errors, where the correct order of the subtree should be [1, 2, 3, 4, 5, 6]. The positioning of the words 6 and 3, relative to the head 5, incurs two instances of head errors since 6 should follow 5 and 3 should precede 5. The words 2, 4, and 1 are on the head's correct side, but their internal order is incorrect. We mark 4 and 1 as sibling errors since the preceding and following words of these two do not have the correct local order. Note that every word can only be involved in at most one type of error since only the dependents without head errors are considered for sibling errors.

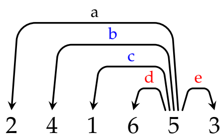


Figure 4.5: An example of two types of error cases, where the correct order should be [1, 2, 3, 4, 5, 6], head errors are marked with red (d and e), sibling errors are marked with blue (b and c).

We first provide the frequency of each dependency relation in each language to focus on analyzing the more frequent relations. The frequency is shown in Figure 4.6, where each row is a language grouped by the language families (solid lines) and subfamilies (dash lines), and each column is a dependency relation grouped by the structural categories (nominals, clauses, modifier words, function words, and the rest).[11] Cells marked with crosses indicate that the treebank has less than 5 cases of the dependency relation.

The relations frequently occur in most languages include the nominal arguments such as nominal subject (`nsubj`), object (`obj`), and oblique (`obl`), modifiers such as nominal modifier (`nmod`), adjectival modifier(`amod`), and adverbial modifier (`advmod`), as well as function words such as determiner (`det`) and case marking (`case`).

We categorize the errors by the dependency relations of the involved words and plot the error rate by the language and dependency relation, similar to the frequency plot. Figure 4.7a and 4.7b illustrate the head errors and sibling errors of the TSP decoder, respectively. We can observe several interesting patterns from the error plots.[12]

scenario, and word ordering in context has more nuances. Therefore the delexicalized and context-free model also does not accurately reflect the word order freedom of a language.

[11] The last group contains all relations that are not dependency relations in the narrow sense. We refer to de Marneffe et al. (2014) for a detailed explanation of the taxonomy of dependency relations.

[12] A caveat to all the following explanations from the error plots is that the patterns are not only influenced

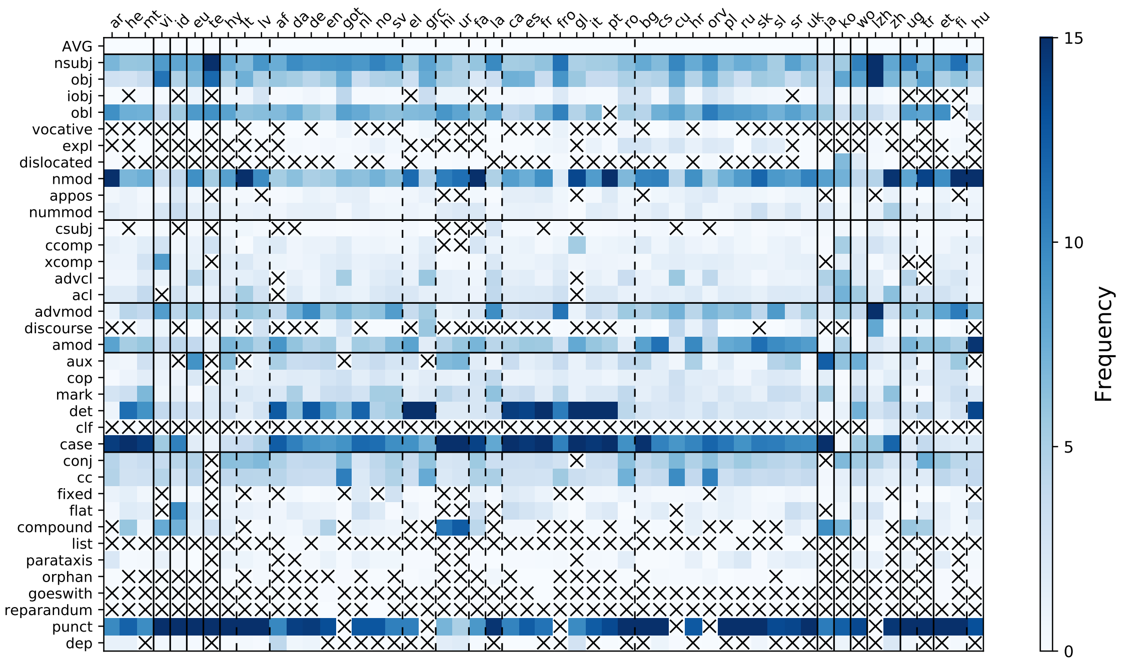


Figure 4.6: Frequency of each dependency relation in each language, where the treebanks are grouped by the language families and subfamilies, and the relations are grouped by the structural categories of the dependent.

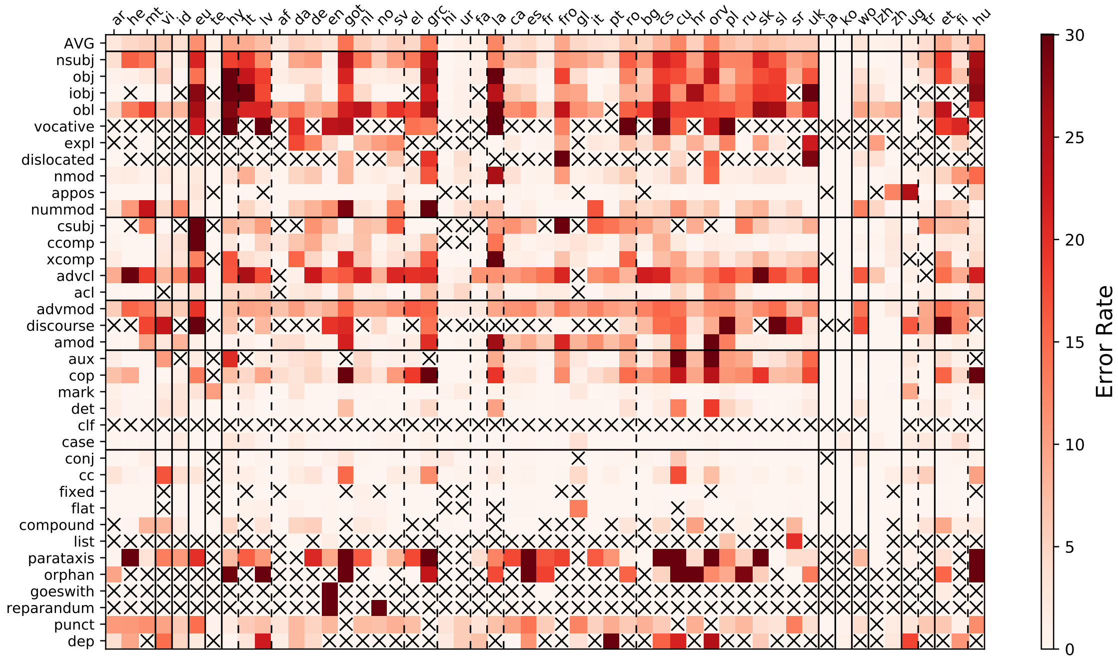
The first major observation is that the two types of errors are mutually supplementary. In other words, the languages that have lower head error rates tend to have higher sibling error rates and vice versa. For example, most of the strictly verb-final languages (hi, ur, fa, ja, ko, ug, tr) have very few head errors because all arguments precede the verb. However, the internal ordering of the arguments is quite free; thus, they tend to have high sibling errors. In contrast, Slavic languages (bg, cs, cu, hr, orv, pl, ru, sk, sl, sr, uk) have high head error rates, but the sibling error rates are in the medium range. Most Romance languages (ca, es, fr, fro, gl, it, pt, ro) have relatively low head and sibling error rates, which suggests that this subfamily has a relatively strict word order. The error characteristics for Germanic languages are more mixed, with German and Dutch having considerably higher sibling errors and Gothic having higher head errors.[13]

When looking at the dimension of dependency relations, the most common errors in both head and siblings occur in the nominal arguments such as nominal subject (`nsubj`),

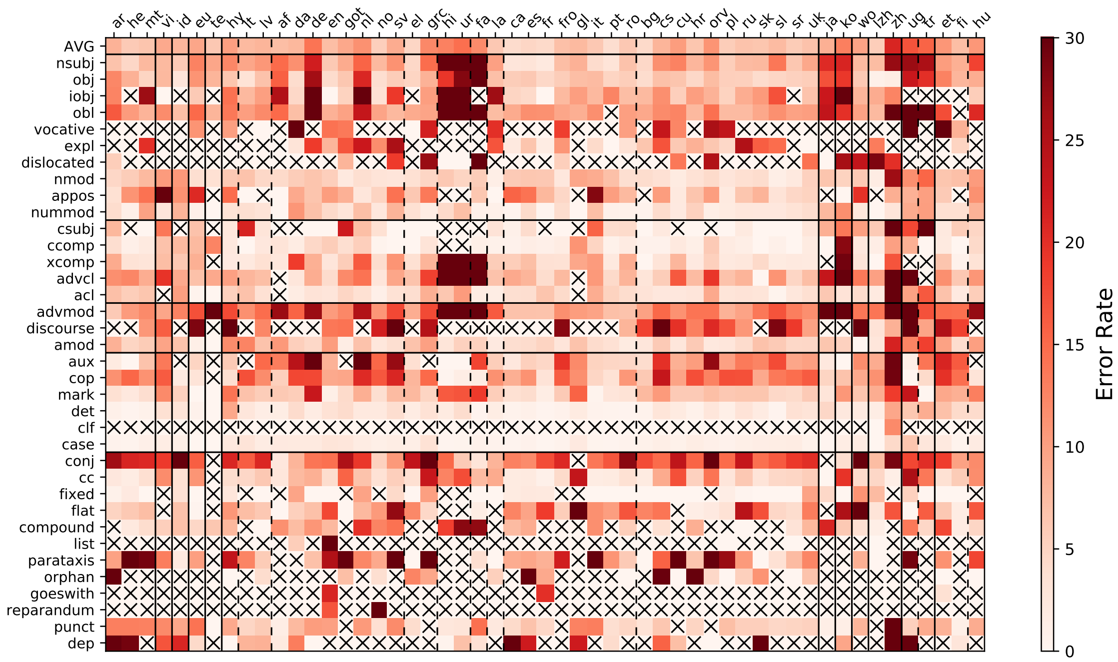
---

by the language itself but also by the genre, annotation idiosyncrasy, and training data size.

[13]The case for Gothic might be related to the genre and annotation scheme of the original treebank since its error pattern is very similar to Ancient Greek, which is also from the PROIEL treebank and contains Bible texts(Haug and Jøhndal, 2008).



(a) head errors



(b) sibling errors

Figure 4.7: Head and sibling error rates of each dependency relation in each language.

oblique (`obl`), object (`obj`), and indirect object (`iobj`). Most function words have relatively fixed order for most languages, except for auxiliary (`aux`) and copula (`cop`) in Slavic languages, as well as some classical Indo-European languages such as Gothic, Ancient Greek, and Latin, which agrees with the observation in the literature that languages tend to obtain more rigid word orders over time (Tily, 2010; Gulordava and Merlo, 2015).

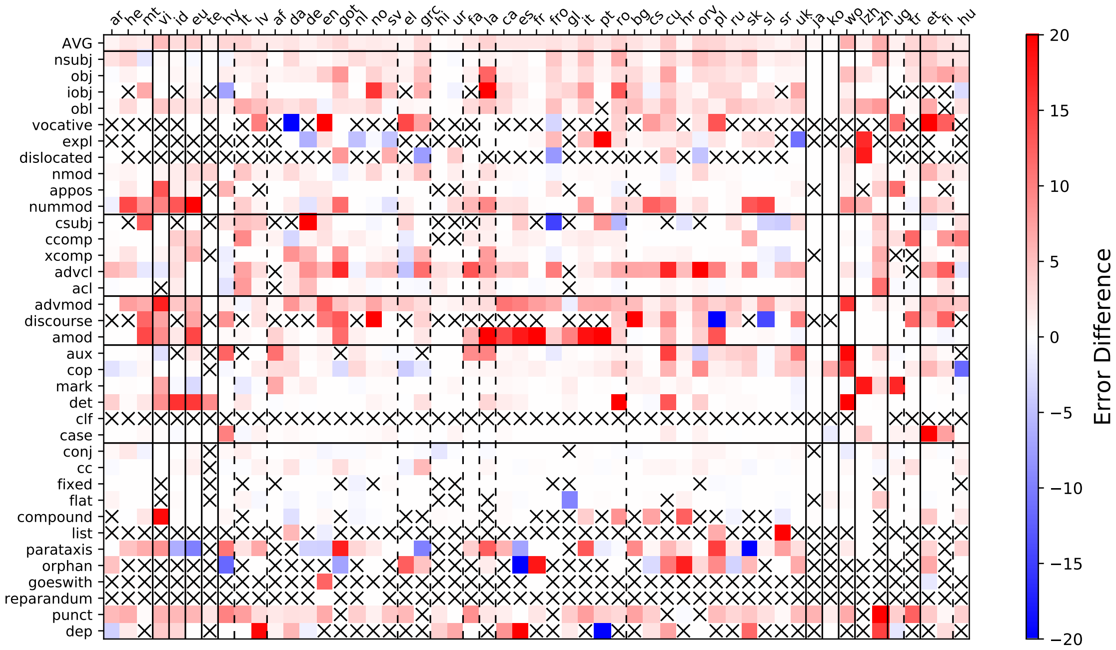
Conjuncts (`conj`) have high sibling error rates for most languages, but they are typically valid alternative orders. Syntactically, conjuncts usually do not prefer a specific order; rather, they are determined by the semantics or world knowledge. For example, *a big red delicious apple* sounds more natural than *a red delicious big apple* due to the hierarchy of adjective types, and *They won the championship in 1999, 2000, and 2001* is better than *They won the championship in 2000, 2001, and 1999* due to the natural sequence of cardinal numbers. The model is trained with too little data to master such nuance. In this respect, a large pre-trained language model could probably achieve better performance.

The previous analyses are conducted on a lexicalized model, i.e., the model has the lexical information of each token. We then compare its errors to a delexicalized model's errors, which tells us what types of dependency relations are more volatile and rely on lexical information to determine the order. The delexicalized model only uses the UPOS and dependency relations as features. Figure 4.8 illustrate the error difference between the normal lexicalized model and the delexicalized model. A red cell indicates that delexicalization causes a drop of performance, in other words, lexicalization is important to determine the order for the corresponding dependency relation. A blue cell is the opposite case, which is quite rare and only occurring in infrequent dependency relations.

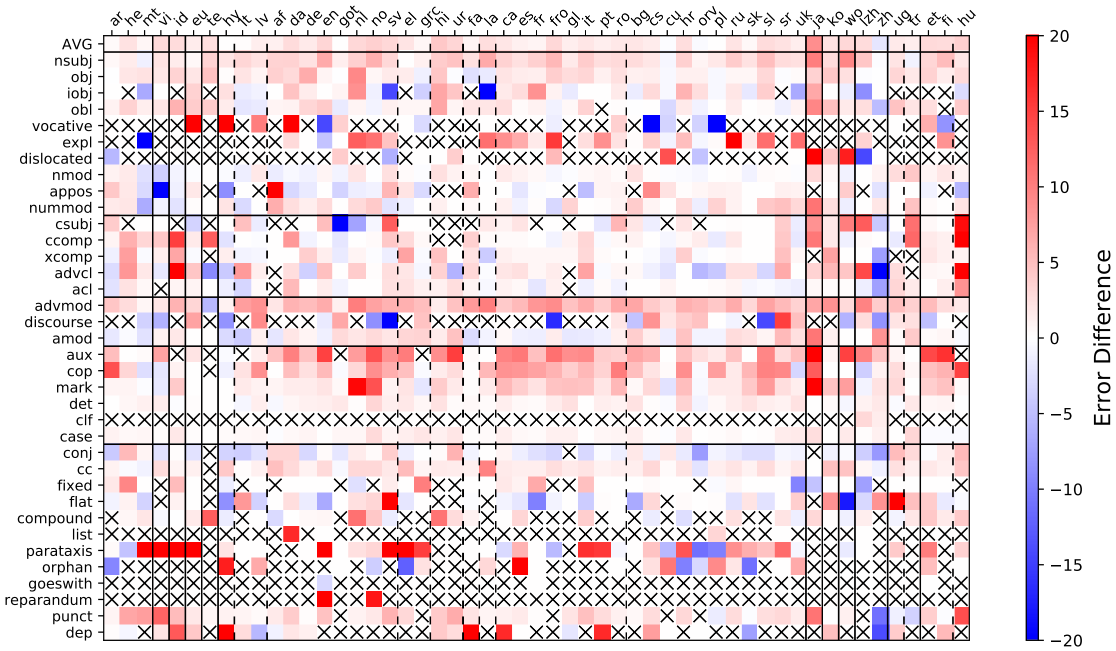
With delexicalization, the most increased error types include the head direction of modifiers (`advmod` and `amod`), especially for Romance languages, where the order between the noun and adjective is usually influenced by their meaning. Although, as discussed before, even lexicalized models could not learn enough semantics from the small data, they can still learn the orders for some frequent words. For example, in Spanish, the adjective can precede or follow the noun, e.g., *un magnífico gol* (a magnificent goal) or *un gol magnífico* are both valid.[14] However, ordinal numbers (classified as adjectives in UD) has strong tendency to precede the noun, such as *la segunda vez* (the second time) instead of *la vez segunda*. However, a delexicalized model could not capture such distinction.

Interestingly, the sibling error rate of conjuncts has no clear change with or without lexicalization, which suggests that its ordering is largely arbitrary or very difficult to learn even for a lexicalized model.

[14] There are subtle differences in the information structure between prenominal and postnominal adjective order, e.g., the prenominal order has +specific reading (Jacob, 2003).



(a) head errors



(b) sibling errors

Figure 4.8: Difference of errors between the lexicalized model and the delexicalized model, which illustrates the benefit of lexicalization.

4.8 Case Studies

We quantitatively analyzed the `TSP-swap` model's errors in the previous section; however, there is no easy way to distinguish actual errors and valid alternative order without qualitatively analyzing the concrete cases. We thus manually evaluate a small subset of the languages, including English, German, and Chinese, for which we have sufficient knowledge to judge the generation quality.

We classify all the predictions deviating from the reference into three categories: (1) **valid** alternative word order or problems with punctuation, (2) **minor** error that can be fixed by swapping a pair of subtrees, and (3) **major** error that cannot be easily fixed. Following are three English examples of the three categories, where in each pair of sentences, the first one is the reference and the second one is the prediction.

(1) a. There 's a Miramar in Florida , just north of Miami .
  b. In Florida just north of Miami , there 's a Miramar .

Although the prediction has quite different order than the reference (and its BLEU score would be 33.52), it is totally grammatical, and should be accepted as an alternative.

(2) a. If you r in the USA and venture into Cuba then u need a passport
  b. Then u need a passport if you r in the USA and venture into Cuba

The prediction is slightly problematic since the clause with *then* should follow the clause with *if*. To fix the problem, one need to simply swap the two clauses (subtrees). Therefore, it is classified as a minor error.

(3) a. I was sold a phone by a friend and sent it off to get it recycled what can i do ?
  b. I was sold a phone what i can do by a friend and sent it off to get it recycled ?

The prediction is also completely scrambled compared to the reference, and it requires several edits to make the sentence readable. Therefore, it is classified as a major error.

Table 4.8 shows the statistics of each deviation category. In general, the distributions of these three categories are quite similar for the three languages. Among the "errors" according to the automatic evaluation, a substantial proportion (30-40%) are valid alternative order; a similarly large proportion (40-50%) are slightly ungrammatical but easy to fix; only less than 20% of the cases are completely wrong. Although we have only evaluated a small subset of languages, this relatively consistent distribution gives an estimation on the extent of real linearization errors.

	English	German	Chinese
valid	39.88%	41.44%	32.46%
minor	46.38%	40.80%	48.54%
major	13.74%	17.76%	19.01%

Table 4.8: Distribution of deviation categories in the prediction.

4.8.1 English

Many real mistakes in English involves interrogative sentences that alter the order of the copula and the subject of a declarative sentence, as in the following sentence.

(4) a. Is R2D2 a stupid name for a cat ?
b. R2D2 is a stupid name for a cat ?

Interestingly, other types of interrogative sentences with a WH-word (*how*, *what*, etc.) or an auxiliary verb (*do*) are always correct. The reason is presumably that the model has an overwhelming preference to produce declarative sentence order, even when the punctuation is a question mark. In contrast, when there is an additional WH-word or auxiliary verb, the declarative sentence order is not possible anymore. Another reason is that the particular English treebank contains a lot of colloquial sentences, where the canonical order of interrogative sentences are not followed, such as the following example. The model then simply learns from the data that even with a question mark, the declarative sentence order is also acceptable.

(5) a. you do n't know what that means ?
b. do n't you know what that means ?

In some cases, the prediction is a grammatical sentence but does not sound natural or has a different meaning than the reference, as in the following examples. In the first pair, the reference sentence means the object *him* can be taken as a summer intern, while the predicted sentence means the subject *I* is the summer intern and can take him. The second pair has almost the same structure and the same meaning shift.

(6) a. I can take him as a summer intern .
b. As a summer intern I can take him .

(7) a. You should treat all pets like children .
b. Like children you should treat all pets .

More commonly, the alternative order is acceptable and has exactly the same meaning as the reference, especially involving temporal and locational modifiers (which all share the dependency label `obl`).

(8) a. It was incorporated as a city on May 26 , 1955 .
b. It was incorporated on May 26 , 1955 as a city .

4.8.2 German

German is a language with relatively high word order freedom, especially regarding verbal arguments. According to the field model (Feldermodell) of German sentence structure (Kürschner, 2017), there are up to five fields for the elements in a sentence: prefield (Vorfeld), left bracket(linke Klammer), midfield(Mittelfeld), right bracket(rechte Klammer), and postfeld (Nachfeld). The finite verb, auxiliary, and copula reside in the left or right brackets, depending on the sentence type. All the arguments and modifiers occupy the prefield or midfield with quite a lot of freedom in terms of their relative ordering.

This flexibility causes the high apparent sibling error rates in the prediction. However, by looking through the concrete cases, we find that many of them are actually valid alternatives, as in the following examples.

(9) a. Ab jetzt verstehe ich die Signale meines Hundes .
from now understand I the signals of-my dog .
b. Ich verstehe ab jetzt die Signale meines Hundes .

(10) a. Es wurde absolut gut und sauber gearbeitet von den Monteuren .
it was absolutely well and clean worked by the assembler .
b. Es wurde von den Monteuren absolut gut und sauber gearbeitet .

There are cases where the subordinate clause and the main clause have the opposite order as in the reference, which is also valid, and the internal structures of the clauses are also correct, as in the following examples.

(11) a. Da wir nichts mehr zu verlieren hatten , versuchten wir es dort .
since we nothing more to lose have , try we it there .
b. Wir versuchten es dort , da wir nichts mehr zu verlieren hatten .

Like the English example, there are also mistakes in German interrogative and imperative sentences, as in the following examples.

(12) a. Kaufen sie bei erfahrenen Züchtern ! !
buy you from experienced breeders
b. Sie kaufen bei erfahrenen Züchtern ! !

(13) a. Wie reimt sich das zusammen ?
how rhyme self it together
b. Wie das reimt sich zusammen ?

4.8.3 Chinese

Chinese is often perceived as a language with a fixed SVO order similar to English. However, the sibling error pattern tells a very different story; all argument types but object have high errors. In the most common constructions, the verb is at the second to last position, and the object is the last, while the other arguments and modifiers are arranged quite freely before or after the subject. Therefore, very few (real or apparent) errors are related to the object since it is likely the only dependent following its head.

The following are examples of alternative arrangements of arguments that have the same meaning as the references.

(14) a. 像 富 人 住宅 一样 , 教堂 和 修道院 通常 也 带有 花园 .
like rich people house alike , church and abbey usually also have garden .
Like the rich people's houses, churches and abbeys also have gardens
b. 教堂 和 修道院 通常 像 富 人 住宅 一样 , 也 带有 花园 .
church and abbey usually like rich people house alike , also have garden .
'Same as above'

(15) a. 随后 爱斯基摩 人 和 维京 人 相继 定居 于 此 .
then Eskimo people and Viking people in-succession settle at here .
'Then the Eskimos and Vikings settled here one after another'
b. 爱斯基摩 人 和 维京 人 随后 于 此 相继 定居 .
Eskimo people and Viking people then at here in-succession settle .
'Same as above'

The following example is a grammatical alternative word order that has a different meaning than the reference. With different placement of the phrase 只 会(can only), it can imply either "only from the cactus and not other plants" or "only absorb nutrient and nothing else".

(16) a. 他们 只 会 从 仙人掌 中 吸取 养分 .
they only can from cactus inside absorb nutrient .
'they can only absorb nutrient from cactus (and not any other plants)'
b. 他们 从 仙人掌 中 只 会 吸取 养分 .
they from cactus inside only can absorb nutrient .
'they can only absorb nutrient from cactus (and absorb nothing else)'

4.9 Summary

In this chapter, we conduct extensive experiments and analyses to determine the best components of the linearization system described in the previous chapter. To encode the input dependency tree, the Tree-LSTM has the overall best performance, while the sequential LSTM and GAT encoders are trailing behind. While comparing to the bag-of-words encoder, it is clear that the syntactic information is indispensable to determine the word order.

We also compare the performance of different decoders. Most decoders have two versions, i.e., on the full tree level and the subtree level. Generally, decoding on the subtree level is always better than the alternative on the tree level since the main problem is recursively broken down into several smaller subproblems, which reduces the search space considerably. Among the subtree-level decoders, the TSP decoder performs significantly higher than the incremental beam search decoders.

One drawback of the subtree-level decoding strategy is that it imposes the projective restriction. To recover the non-projective sentences, we use a transition system to reorder the projective sentences as a post-processing step. The experiment shows that it consistently improves the performance on non-projective sentences for almost all languages with minimal negative impact on projective sentences.

Through the experiments, we have established that the optimal linearization model in our framework consists of (1) a Tree-LSTM encoder to represent the dependency tree and (2) a TSP decoder to order each subtree, and (3) a transition system to reorder the resulting projective sentence.

Since the TSP decoder uses a greedy solver to search for the optimal sequence given the bigram scores from the machine learning model, it is of interest to understand where the errors originate from so that we could direct further efforts for optimization. We measure the frequencies of model errors and search errors, and conclude that the the subtree-level TSP decoder incur very few search errors, thus justifying the choice of the greedy solver for its speed advantages.

To test the data efficiency of our model, we also control the training data size and experiment with low-resource scenarios. The performance decline with smaller training data size is smooth, i.e., without abrupt drop, which indicates that our model is a practical choice for extremely low-resource languages.

We then make detailed analyses of the cases where our model's prediction deviates from the reference in terms of dependency relations and language families. We break down the word order errors into two types, a head error is when a dependent is on the wrong side of its head, and a sibling error is when a dependent on the correct side of the head but the wrong side of its direct neighbors. We observe that most languages have either a high head error rate (Slavic languages) or a high sibling error rate (most verb-final languages). Romance languages have relatively low error rates, only a few languages have high error rates in both aspects. We then compare the difference of errors between the lexicalized and delexicalized models. The results suggest that lexicalization helps mostly resolve the order regarding adjectival modifiers and some function words such as auxiliary and copula.

It is important to note that the errors in the predictions compared to the references are not necessarily incorrect; they are just different from the single reference sentence in the treebank. In many cases, the predictions are valid alternative arrangements. Therefore, we manually inspect a few languages to distinguish actual linearization errors from word order variations. Indeed, more than one-third of the error cases have perfectly grammatical alternative order. The most common alternative order involves the arguments such as subject, object, and oblique. Conjunct is another example where any ordering is syntactically possible: only in some cases would the semantics indicate a preferred ordering, which is also very difficult to learn with a limited amount of data.

Chapter 5

Dependency Length Minimization

In the previous chapter, we have analyzed the errors of the linearization model and demonstrated with examples that many of the apparent errors are valid alternative orders. Indeed, the error rate can be viewed as a proxy of the flexibility of word order. In this chapter, we try to understand the freedom of word order variations from a typological linguistic perspective. In particular, we explore a hypothesis that universally influences the word order in all languages, namely the Dependency Length Minimization (DLM) hypothesis. Phrased as a language typological universal, it states that the evolution of languages is driven by the constraint that grammars should allow the dependents to be realized as closely as possible to their heads – which is known to reduce the cognitive burden in processing (Gibson, 1998, 2000).

The Dependency Length (DL) of a sentence is defined as the sum of the distance between the head and dependent of all the dependency arcs in the sentence (see the example in Figure 5.1). The DLM hypothesis predicts that when different ways to arrange the dependents are possible, we would prefer the one with a shorter DL. In this example, we would be thus more likely to say *John threw out the trash sitting in the kitchen* instead of *John threw the trash sitting in the kitchen out*.

We first conduct experiments to confirm the influence of DLM by comparing the DL between the actual sentences from a treebank and their random ordering under different constraints (which are called baselines in the literature). Through the design of different baselines, we can isolate different types of word order constraints, such as head direction, sibling arrangement, and projectivity, and demonstrate that DLM manifests itself in each one of these constraints.

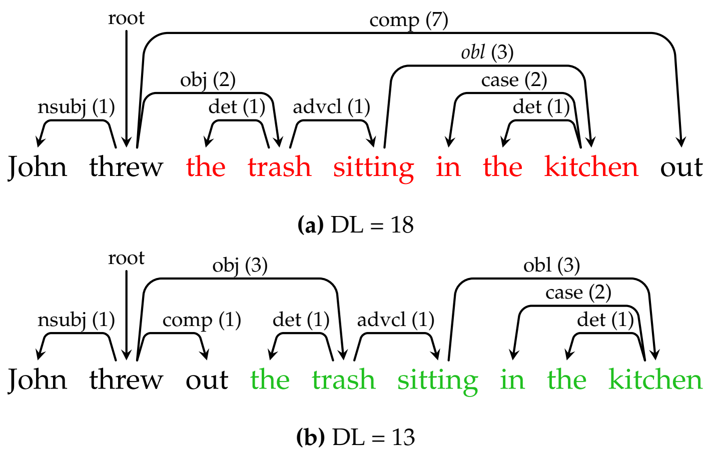


Figure 5.1: Example dependency trees and their dependency lengths, adapted from Futrell et al. (2015a). The second sentence is preferred since it has shorter DL and lower cognitive burden.

5.1 Experimental Settings

5.1.1 Data

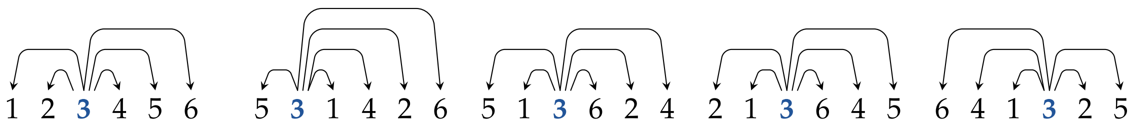
We conduct our experiments on a selection of 53 treebanks from Universal Dependencies v2.4 (Nivre et al., 2019), same as in Chapter 4. The selection consists of training sets from all treebanks with at least 500 sentences. When there are multiple treebanks for one language, we use the largest lexicalized treebank to ensure stable estimation. We remove punctuation from trees and do not consider it when calculating DL since punctuation biases statistics by introducing many long-distance right dependencies, and such dependencies contribute very little to the meaning of the structure.[15]

First, we only consider projective trees since the non-projectivity could impact the DL in other ways, as shown in Ferrer-i-Cancho and Gómez-Rodríguez (2016). Focusing on projective trees allows us to (1) analyze the DL on the subtree level since the internal ordering of each subtree does not influence other subtrees; and (2) efficiently find the optimal ordering in terms of DL (Gildea and Temperley, 2007). Later, we turn our focus to the non-projective trees and analyze the relation between non-projectivity and DL.

5.1.2 Baselines

We compare the DL of the observed sentences in the treebanks (OBS) with four baselines that generate random linearizations with incremental constraints toward the actual data:

[15]The role of most punctuation marks is to segment the phrases or clauses, which is a part of the surface order but not relevant to the structural order.



(a) OBS **(b)** FREE **(c)** VAL **(d)** SIDE **(e)** OPT

Figure 5.2: An example tree and results of four baseline linearization methods, where the label represents the size of the subtree.

FREE: free word order baseline, which does not impose any constraints except projectivity to the linearization. It is also used in Futrell et al. (2015a).

VAL: same-valency baseline, which ensures that the numbers of left and right dependents of a head in the random subtree are the same as in the real data.[16] In other words, we shuffle the left and right dependents in the original tree separately.

SIDE: same-side baseline, which ensures that the dependents in the random tree stay on the same side of the head as in the real data. This also satisfies the constraints in VAL and differs from OBS only in the linear arrangement of dependents on each side.[17]

OPT: optimal baseline, which minimizes the dependency length, as in Gildea and Temperley (2007). The basic idea is to sort the dependents by their subtree sizes and attach them to the left and right sides of the head alternately.

Figure 5.2 illustrates and example for each baseline, where (a) is the original ordering; (b) shuffles all the tokens in the tree; (c) ensures that there are 2 left dependents and 3 right dependents as in (a); (d) shuffles the left and right dependents of (a) separately; and (e) is the optimal ordering of the tokens, assuming the label of each dependent also signifies the size of its subtree.

5.1.3 Baseline Dependency Lengths

Figure 5.3 illustrates the average DL of the five baselines with respect to the sentence length.[18] From the longest to the shortest are: FREE, VAL, SIDE, OBS, and OPT, and each adjacent pair is clearly separated. By gradually imposing word order constraints extracted from the actual data, assimilating a random tree toward the actual tree, the average DL will keep reducing.

In the following analyses, we make pair-wise comparisons of the baselines to demon-

[16] We use the term *valency* only to describe the number of dependents, not their types.

[17] SIDE baseline is mentioned but not analyzed in the appendix of Futrell et al. (2015a).

[18] The general trend and ranking of the baselines in each individual treebank are consistent with the average.

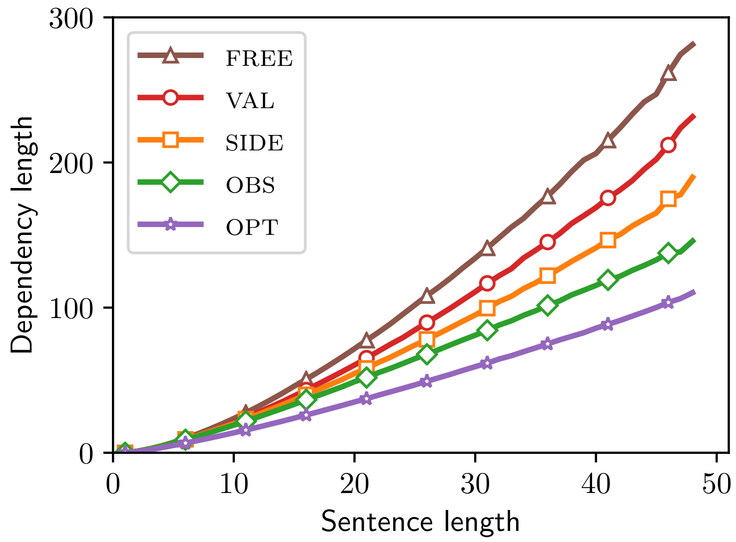


Figure 5.3: Average DL vs. sentence length across 53 treebanks.

strate the different types of word order constraints that separate the pair, and explain how they contribute to the DL reduction.

§5.2.2 explains the influence of balanced left and right dependents to the DL (FREE vs. VAL); §5.2.3 illustrates the influence of the head direction as a word order constraint (VAL vs. SIDE); and §5.2.4 shows the ordering of same-side siblings as another word order constraint (SIDE vs. OBS). In each of the aspects, the constraints from the observed data show a clear preference for shorter DL but do not reach the shortest possible DL as in OPT, since DLM is not the only force that shapes the word order.

5.2 Analyses

5.2.1 Word Order Constraints

Before comparing the baselines, we first introduce two types of word order constraints, the **head direction** and the **sibling ordering**, which we will use as metrics to quantify the freedom of word order and as tunable parameters to manipulate the tree.

The head direction constraint describes on which side of the head lies the dependent for each dependency pattern. Its dependency pattern consists of the UPOS of the head, the UPOS of the dependent, and the dependency label. An example dependency pattern from Figure 5.1 is <VERB, ADP, comp>, and its value is *right*.

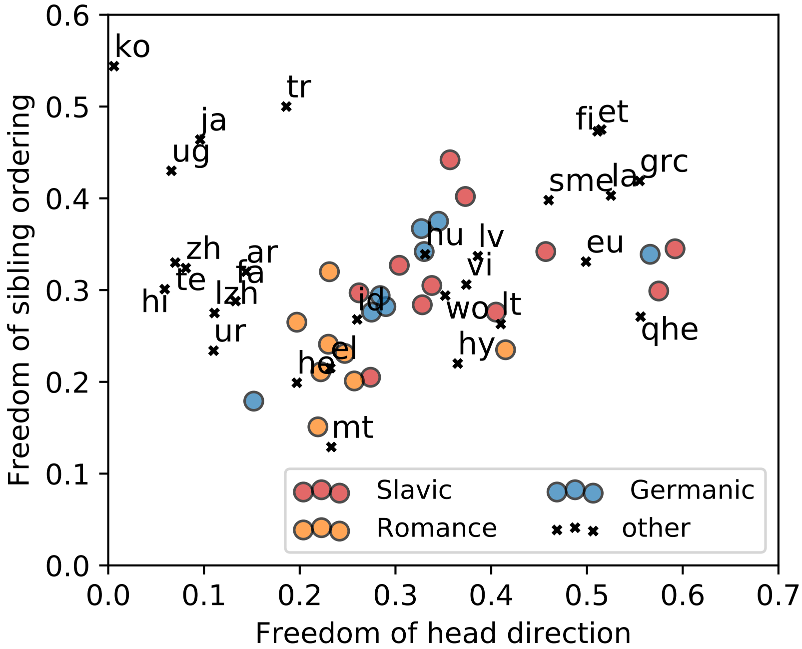


Figure 5.4: All treebanks characterized by the two types of word order freedom.

The sibling ordering constraint describes the pairwise precedence relation of the dependents on the same side of the head. It is an approximation of the total order of the siblings, but much more straightforward and resistant to the data scarcity problem (Gulordava, 2018). Its dependency pattern consists of the UPOS of the head, the side of the involved dependents, and the UPOS and label of the two dependents. An example is <VERB, right, ADP, comp, NOUN, obj>, and the value is *left* for Figure 5.1b and *right* for Figure 5.1a.

For both dependency patterns, we count the frequency of their values and use the entropy to measure the **freedom** of that pattern.

$$E = -\sum_i p_i \log p_i \tag{5.1}$$

For a variable with two possible values (left and right), the entropy ranges from 0 (always the same value) to 1 (both values equally likely). For example, if a noun appears 50 times on the left of a verb as the subject, and 10 times on the right, then the entropy of <VERB, NOUN, obj> is 0.65.

We then measure the freedom of the overall word order constraints by taking the average entropy of each single dependency pattern in that constraint type weighted by its frequency in the data. Figure 5.4 shows all treebanks characterized by the two types of word order freedom. We mark three most common language subfamilies and annotate the rest with their language codes.

Generally, we do not see any strong correlation of these two types of freedom, which indicates that they characterize different aspects of the word order. Many verb-final languages cluster near the top left corner since they tend to have strict head direction constraints that all arguments of a verb are uttered before the verb, but the exact ordering of the arguments is very flexible.

The characteristics of these two types of constraints are pretty similar to the two types of errors defined in §4.7. They both break down the word order in two aspects, head direction and sibling arrangement. The difference is that we use the constraints to describe the word order freedom within the original treebank, and we use the errors to analyze the deviation between the original sentence and the model prediction.

The constraints can be used not only to reflect the freedom, but also as a parameter that we can manipulate, as we will show in §5.2.3 and §5.2.4. They are more fine-grained (with combinations of UPOS and dependency label), thus difficult to visualize, and we only report them on the average level. The error types are used to reflect the confusion of the linearization model, and we use a more coarse description (with only dependency label) for the sake of visualization.

5.2.2 Dependent Distribution: FREE vs. VAL

The first pair of baselines that we compare in Figure 5.3 is FREE and VAL. FREE is a random arrangement of each subtree without any local constraints, while VAL imposes the valency constraints on the FREE baseline, which results in the first reduction of DL. It can be easily explained by the fact that DL is shorter when the left and right dependents are more balanced, or, in other words, when the head is positioned in the middle of its dependents, as shown in Ferrer-i-Cancho (2015). In the FREE baseline, the head is equally likely to be placed in any position of the dependents, while in the VAL baseline, where the number of left and right dependents of each head is the same as from the actual tree, it is more likely than chance to have a balanced number of dependents on both sides.

To demonstrate this fact, we measure the **imbalance** of a head by the difference of numbers of dependents on both sides divided by the number of dependents. The more balanced a tree is, the smaller the value. We take the averaged imbalance value of all heads with more than one dependent as the imbalance measurement of the whole treebank. We also calculate the expected imbalance of FREE. For a head with n dependents, there are $n + 1$ possible location to insert the head, and the difference of dependents at each location would be: $\{n, n-2, ..., 2, 0, 2, ..., n-2, n\}$ if n is even, or $\{n, n-2, ..., 1, 1, ..., n-2, n\}$ if n is odd. The sum of these values is $\lfloor\frac{(n+1)^2}{2}\rfloor$, normalized by the length n and averaged by the equiprobable locations $n+1$, thus the expected imbalance value is:

$$imb(n) = \lfloor\frac{(n+1)^2}{2}\rfloor/n(n+1) \tag{5.2}$$

The measured average imbalance value for VAL is 0.47 (same as OBS) and 0.67 for FREE (also very close to the expectation from the formula) , which means that VAL distributes the dependents in a more balanced way. There are only three languages (Telugu, Uyghur, and Korean) where VAL is more imbalanced, mainly because they are verb-final languages, where all the arguments are on one side of the verb. This general trend indicates that most natural languages tend to have more balanced dependents than chance level (in other words, position the head at a more central position), thus reducing of DL from FREE to VAL.

Note that in this scenario, we only consider the *number* of dependents as the measurement of balance, which is a simplistic heuristics, while the *subtree size* of the dependents is the more accurate measurement. We explore this factor in §5.2.4.

5.2.3 Head Direction: VAL vs. SIDE

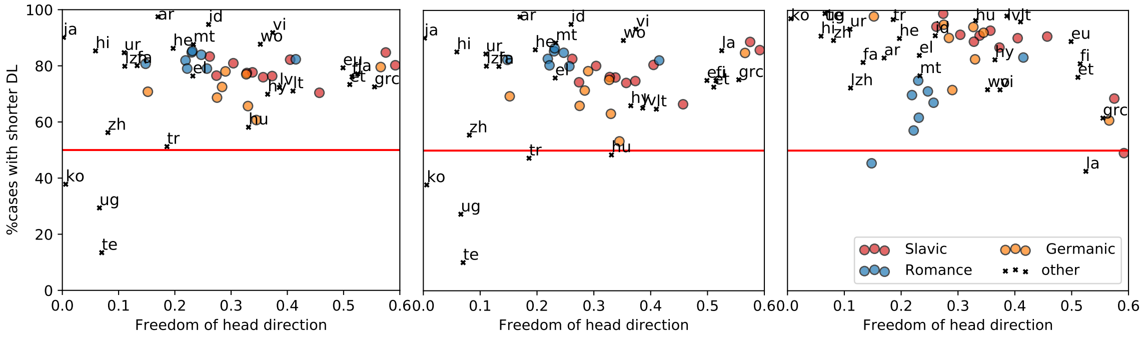
Next, we consider the length reduction from VAL to SIDE, which puts additional constraints on the head direction based on the actual data.

We illustrate the relation between the head direction preference and its effect on DL through two scenarios. The first scenario studies the regularity of head direction: we flip the dependent to the other side of the head if it is on the **majority** side of its dependency pattern based on the statistics of the actual data. The second scenario studies the exception of head direction: we flip the dependent if it is on the **minority** side.

To put it differently, the majority side includes cases reflecting the preference of a dependency pattern, such as the typical adjective-noun order in English; the minority side includes the exceptions of the dependency pattern, such as the unusual noun-adjective order in *the bear and the maiden fair*.[19]

While flipping the dependent, we keep the order of all other dependents unchanged and insert the flipped dependent into a position that minimizes the DL. This way, we make sure that the hypothetical flipping does not unnecessarily increase the DL, and the comparison to the original order is fair.

[19] https://en.wikipedia.org/wiki/The_Bear_and_the_Maiden_Fair_(song)



(a) overall **(b)** majority scenario **(c)** minority scenario

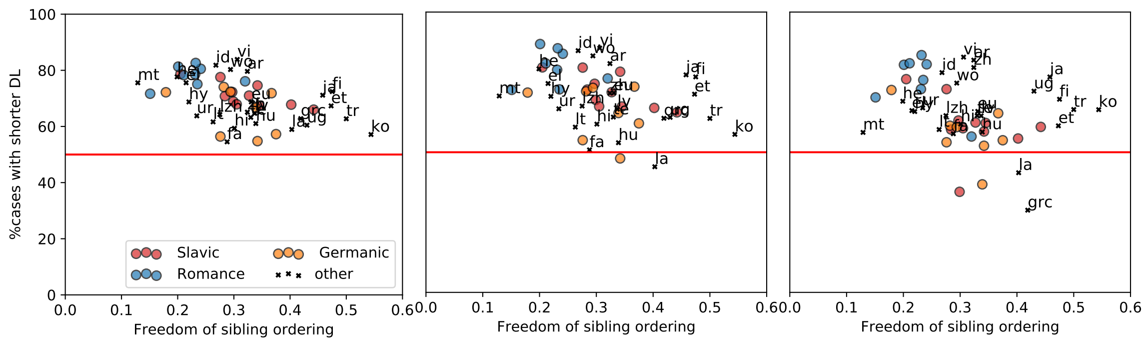
Figure 5.5: Percentage of trees where the real order has shorter DL than flipping one dependent.

The results are shown in Figure 5.5. The y-axis shows the percentage of cases where the original order has shorter DL, i.e., flipping would increase the DL. The x-axis shows the freedom of head direction. If a point lies above the 50% line (red line on the plot), it means that the actual data in that treebank has a shorter DL than by chance.

Figure 5.5a shows the overall trend of the flipping experiment, where the actual data is more likely to have shorter DL than the flipped ones, except for a few verb-final languages (Korean, Japanese, Telugu, and Uyghur), similar to the exceptions in §5.2.2. Since they tend to utter all the arguments before the verb, flipping any arguments would balance the head and thus reduce the DL.

Note that by flipping one dependent to the other side, the valency of the head is also changed; therefore, it is possible that the reduction of DL is caused by the effect described in §5.2.2. However, we record the imbalance before and after the flipping, and it is almost not changed on average. In other words, the influence from the change of valency is very small for this experiment.

Figure 5.5b shows the majority flipping scenario, which is very similar to the overall picture, since this scenario has more test cases (by definition), therefore dominates the statistics. The plot shows that the canonical order leads to shorter DL for most of the languages, thus supports the hypothesis that DLM drives the evolution of word order. This then invites the question of why we still use non-canonical order in utterance. We can explain it in terms of DLM by observing the minority scenario, where we "correct" the head direction in the actual data if it is not the majority choice. As shown in Figure 5.5c, for almost all languages, the original minority order has a shorter DL than the flipped majority order. More generally, the minority scenario results resemble the mirror image of the majority case, namely, the languages with a shorter DL in the majority case would have less a shorter DL in the minority case.



(a) overall **(b)** majority scenario **(c)** minority scenario

Figure 5.6: Percentage of trees where the real order has shorter DL than swapping two siblings.

Assuming the head direction in each instance is realized without the influence of DL, then either the majority or the minority scenario should have less than 50% cases with shorter DL. However, the fact that both scenarios have more than 50% positive cases indicates that DLM indeed influences the shaping of the head direction preferences as well as motivating the deviation from such preferences.

5.2.4 Sibling Ordering: SIDE vs. OBS

Finally, we look at the DL reduction from SIDE to OBS, which only differ in the ordering of siblings on the same side. Their difference in DL indicates that the ordering of the same-side siblings is also influenced by DLM, as in the motivating example in Figure 5.1. According to the explanation of DLM, we prefer (b) over (a) because when both variants are grammatical, we tend to keep smaller subtrees closer to the head helps to reduce DL and cognitive effort.

To verify whether the data supports the claim, we conduct similar experiments as in §5.2.3, where we compare every subtree in the original data to the modification where we swap one pair of dependents on the same side. We also compare two scenarios, majority and minority, where the swapped pair belongs to the regularity and exception of its dependency pattern, respectively.

The results are shown in Figure 5.6. We notice, that overall as well as in both majority and minority scenarios, the original subtrees have shorter DL than swapping two dependents. This again indicates that given the head direction, the natural language tends to arrange the dependents in a way to minimize the DL, regardless of whether the order is predominant or not. It also supports the similar conclusion as in §5.2.3 that DLM motivates both the regularity and exceptions in the ordering of siblings.

5.3 Experiments with Linearizer

We now include the TSP-based linearizer as described in Chapter 3. To ensure a fair comparison, we do not use the lexical information as features. Two models with different features are used: the first model (DELEX) only uses UPOS tags and dependency relations, same as the baselines in this chapter; the second model (STRUCT) only perceive the topological structure of the tree and nothing else, in which we use the same dummy vector to represent all tokens, and the only useful information about the input is the tree structure (e.g. the size of each subtree) learned by the Tree-LSTM.

We conducted the baseline experiments on the training set for more stable statistics, since the training set is much larger than the development set. Therefore, we also test our linearization model on the training set with 2-fold jackknifing. Concretely, we break the training set into two halves and train two models on each half, then use each model to predict the other half of the training set. After the prediction, we remove the punctuation to calculate the DL.

Figure 5.7 compares the DLs between the original sentences and the predicted linearization. It includes the baselines from Figure 5.3 as references, with two additional data series: the first line (DELEX) is the prediction of the delexicalized TSP model, and the second line (STRUCT) is from the structure-only model.

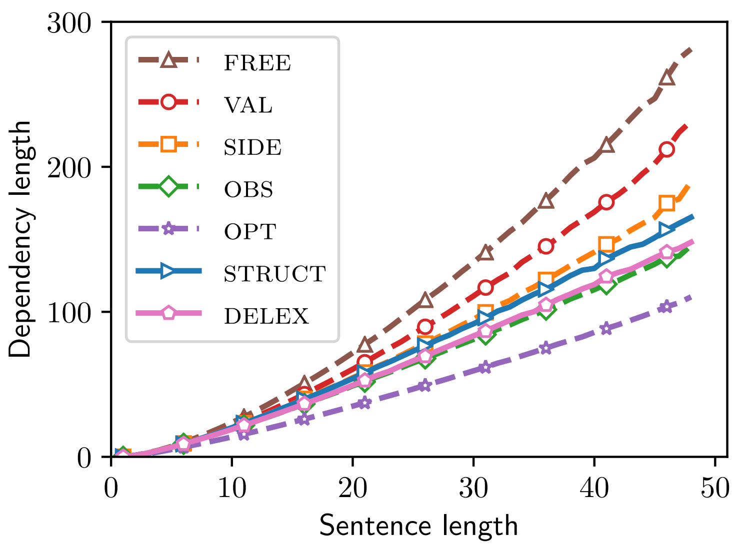


Figure 5.7: Average DL vs. sentence length across 53 treebanks. The additional results of DELEX and STRUCT are from the trained linearization models, while the others are directly from manipulation of the original treebank.

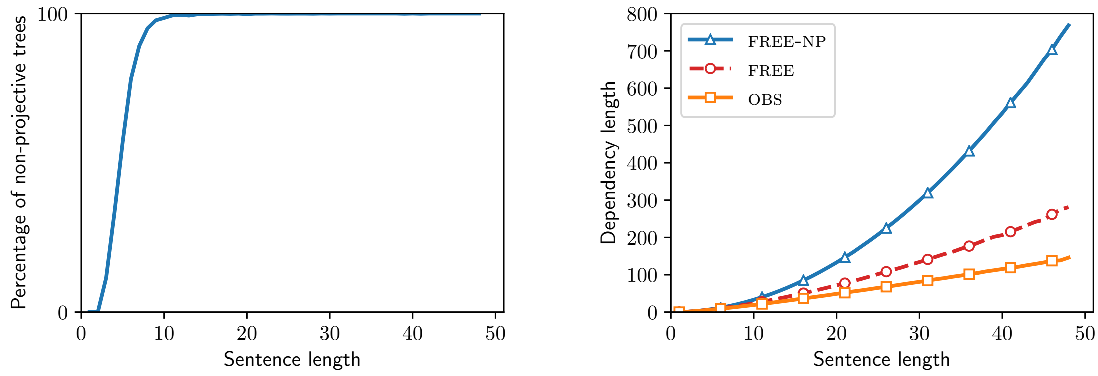
Generally, the difference in terms of DL between the original data (OBS) and the delexicalized linearization model (DELEX) is almost indistinguishable, which means that the model has already implicitly factored in the DL while making the prediction. More interestingly, the much weaker, structure-only linearization model (STRUCT) also achieves quite small DL very close to OBS, and clearly smaller than the SIDE baseline, which differs from OBS only in the that it does not take the subtree size into account when arranging the siblings. The fact that STRUCT has shorter DL than SIDE suggests that the model to some extent learns to arrange smaller subtrees closer to the head, although not as good as the DELEX model. This suggests that the Tree-LSTM is able to learn the topological characteristics of the typical sentences and optimize the DL to an impressive level.

5.4 DLM and Non-Projectivity

In the previous experiments, we largely follow the setting in Futrell et al. (2015a) which excludes non-projective sentences while analyzing the effect of DLM. There has been a line of work that explores the relation between DLM and non-projectivity, in particular, the scarcity of non-projectivity in natural language (Ferrer-i-Cancho and Gómez-Rodríguez, 2016; Gómez-Rodríguez and Ferrer-i-Cancho, 2017). If the words in a dependency tree are ordered at random, the probability of a non-projective tree would be overwhelmingly high. However, in the actual data, very few sentences are non-projective. The authors find that the DLM hypothesis could explain this phenomenon.

Similar to their work, we first simulate the fully random trees that are not constrained by projectivity. We shuffle the word order of all the sentences in the training set and record the percentage of non-projective sentences as well as the DL with respect to the sentence lengths. As shown in Figure 5.8a, a fully random tree is very likely to be non-projective. With a length of more than 10, the probability is close to 100%. Figure 5.8a shows the average DL of the fully random tree (FREE-NP), compared to the projective random tree (FREE) and the actual trees (OBS). Clearly, the fully random trees are significantly higher than any other baselines.

This observation provides an evidence for the role of DLM in suppressing non-projectivity in natural languages. However, one would ask whether the existence of the few non-projective trees could also be explained by DLM. Intuitively, it could well be the case. Take the previous German example with an extraposed relative clause, as shown in Figure 5.9. The projective order in (a) is also acceptable (ignoring the issue of a missing comma), and appears more frequently in written text than in colloquial speech, since we



(a) percentage of random non-projective trees

(b) dependency lengths

Figure 5.8: The percentage of non-projective trees and the DL of the fully random linearization, compared to the projective random linearization and the real data.

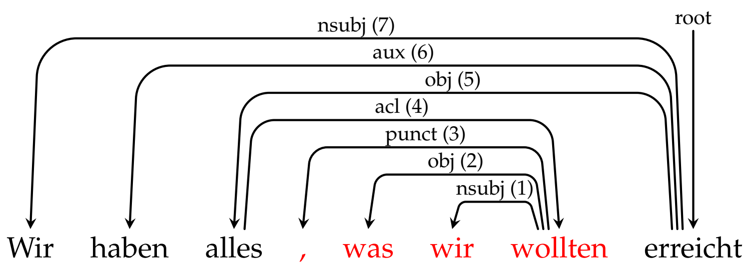
have higher cognitive capacity for written text.[20] In colloquial speech, the relative clause is more likely to be moved outside of the main clause, since one cannot keep the arguments in the memory for too long before the main verb appears. AS a result, one might either forget to say the main verb (as the speaker) or forget the context (as the listener). By extraposing the relative clause, thus creating a non-projective arc, the DL of the sentence could be drastically reduced, from 28 to 17 in the example.

We then proceed to to verify this hypothesis on a large scale. We collect all the non-projective trees in the treebanks and projectivize them by keeping the local subtree ordering. In this way, we can isolate the (non-)projectivity from other word order constraints in the analysis.

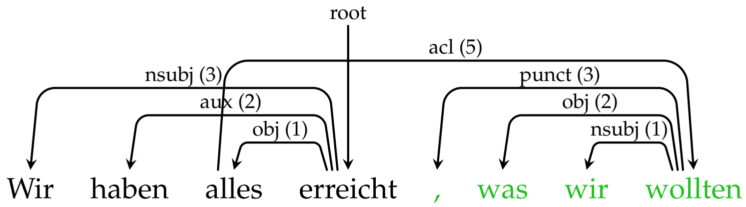
We compare the change of DL between the non-projective and the projectivized sentences, as shown in Figure 5.10, where (a) shows the average reduction of DL (normalized by the sentence length) in the non-projective sentences compared to the projectivized ones, and (b) shows the percentage of non-projective sentences with shorter DL than its projectivized version. In both figures, the red line represents the null-hypothesis that there is no difference between the non-projective and projectivized sentences. To reduce the influence of the annotation scheme, we only consider the pairs with a normalized DL difference over 0.5, e.g., the example in Figure 5.9 has a differences of $(28-17)/8 = 1.375$.

For the majority of languages, the non-projective sentences have shorter DL than the projectivized sentences both in terms of average and percentage of cases. This suggests that the non-projectivity in natural language could also be driven by DLM.

[20]It remains a great obstacle for learners, as depicted in Mark Twain's essay *The Awful German Language*.

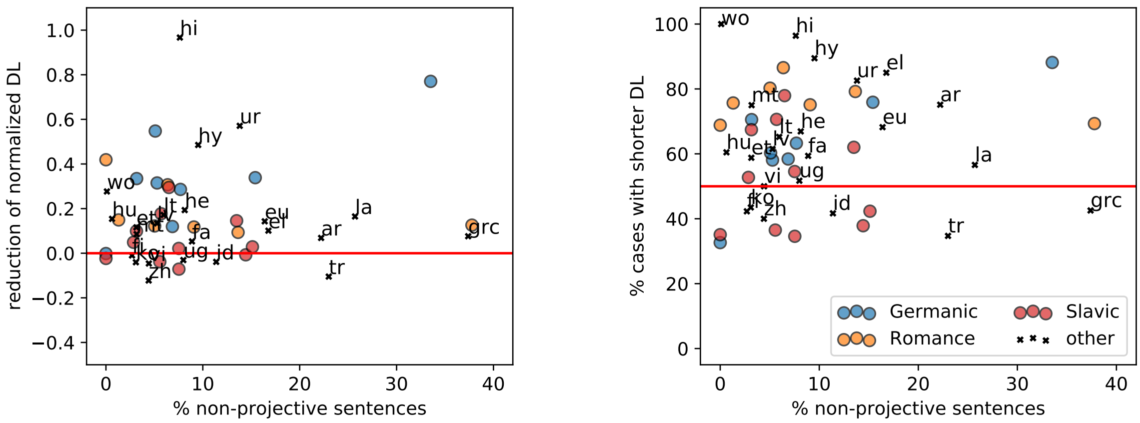


(a) DL = 28



(b) DL = 17

Figure 5.9: A pair of projective and non-projective sentence in German, and their DLs. The non-projective sentence with the extraposed relative clause has shorter DL.



(a) reduction of normalized DL in NP sentences **(b)** percentage of NP sentences with shorter DL

Figure 5.10: The comparisons between the non-projective sentences and their projectivized versions in terms of the DL.

Combining the two findings regarding non-projectivity, we could reach a conclusion similar to the other local word order constraints; that is, the DLM motivates the lack of non-projectivity as well as the existence thereof.

5.5 Summary

This chapter builds on the basis of the experimental setting and analysis in Futrell et al. (2015a), which illustrates the effect of DLM by comparing the actual data with random linearization of the dependency tree.

We break down the effect of DLM step by step and analyze its relation to the local constraints such as dependent distribution, head direction, and sibling arrangement. The systematic breakdown indicates that natural languages universally show a clear preference for shorter dependency length in all three aspects, both in the regularities and in the exceptions of word orders. Our findings provide more detailed evidence for the hypothesis that DLM is a universal phenomenon in natural language.

Apart from the baselines that are based on the original trees, we also apply our linearization model for comparison. The result shows that the prediction from a delexicalized model has a very similar DL to the original sentence, which suggests that the model has already implicitly learned the DLM principle. Moreover, we apply a much simpler model that is entirely agnostic of the syntax and semantics of the sentence; instead, it is only aware of the tree structure. The prediction of this model has only slightly longer DL than the actual data, much shorter than the random projective baseline.

Finally, we explore the last puzzle piece regarding word order, namely the global constraint of projectivity. The simulation shows that the fully random word order has a very high probability of introducing non-projective arcs, and they would, in turn, significantly increase the DL compared to a random projective tree. This result agrees with the theoretical work by Ferrer-i-Cancho (2006) that the scarcity of non-projective trees could be a side effect of DLM. Furthermore, we explore the question from the opposite direction, i.e., whether the existing non-projective is also influenced by DLM. The comparison between the real non-projective trees and their hypothetical projectivized tree shows that indeed in the majority of languages, the non-projective trees have shorter DLs.

Through our step-wise experiments, we have demonstrated that the DLM plays an essential role in every aspect of word order, including the direction of the head and the arrangement of siblings at the local level, as well as the projectivity at the global level. It shapes the general word order preferences, and it also motivates the deviation from the preferences in certain utterances. It is undoubtedly desirable for linearization models to be aware of such an important factor, and we have seen that our model is indeed capable of following the DLM principle.

Chapter 6

Surface Realization Shared Tasks

6.1 Task Description

Surface realization is the task of generating a well-formed sentence given a structured meaning representation. A series of shared tasks have been organized that aim to generating sentences from unordered dependency trees. It begins with the first edition (Belz et al., 2011), where only one English dataset is used. Later, with the development of the UD, a multilingual version of the shared task has been organized in consecutive years (Mille et al., 2018, 2019, 2020), in which up to 11 languages from the UD are tested.

The shared tasks always have two tracks, the shallow track (T1) and the deep track (T2). In the shallow track, the input is an unordered dependency tree that includes all the tokens in the output sentence, the same setting we use in Chapter 3. In the deep track, the input only contains content words, while the function words such as prepositions, auxiliaries, and punctuation, are removed by a rule-based script from the organizers. This setting aims to simulate a more realistic scenario, which is closer to a semantics-oriented meaning representation.

We participated in the shared task in 2019 and 2020, referred to as SR'19 and SR'20, respectively. Table 6.1 summarizes the treebanks grouped in different scenarios. The in-domain group includes the original UD treebanks with train/dev/test splits; the out-of-domain group includes treebanks with only test sets; the automatically parsed group includes a subset of the in-domain treebanks parsed by the best performing parser of each language from the CoNLL'18 shared task (Zeman et al., 2018). A new group is introduced in SR'20, which includes Wikipedia texts parsed by the Stanza parser (Qi et al., 2020). Among the 11 languages, only English, French, and Spanish are used in the deep track, the rest only in the shallow track.

Data type	Dataset	Track	train	dev	test
In-domain	ar_padt	T1	6,075	909	680
	en_ewt	T1, T2	12,543	2,002	2,077
	en_gum	T1, T2	2,914	707	778
	en_lines	T1, T2	2,738	912	914
	en_partut	T1, T2	1,781	156	153
	es_ancora	T1, T2	14,305	1,654	1,721
	es_gsd	T1, T2	14,187	1,400	426
	fr_gsd	T1, T2	14,450	1,476	416
	fr_partut	T1, T2	803	107	110
	fr_sequoia	T1, T2	2,231	412	456
	hi_hdtb	T1	13,304	1,659	1,684
	id_gsd	T1	4,477	559	557
	ja_gsd	T1	7,133	511	551
	ko_gsd	T1	4,400	950	989
	ko_kaist	T1	23,010	2,066	2,287
	pt_bosque	T1	8,328	560	477
	pt_gsd	T1	9,664	1,210	1,204
	ru_gsd	T1	3,850	579	601
	ru_syntagrus	T1	48,814	6,584	6,491
	zh_gsd	T1	3,997	500	500
Out-of-domain	en_pud	T1, T2	-	-	1,000
	ja_pud	T1	-	-	1,000
	ru_pud	T1	-	-	1,000
Automatically parsed	en_ewt-pred	T1, T2	-	-	1,795
	en_pud-pred	T1, T2	-	-	1,032
	es_ancora-pred	T1, T2	-	-	1,723
	hi_hdtb-pred	T1	-	-	1,675
	ko_kaist-pred	T1	-	-	2,287
	pt_bosque-pred	T1	-	-	471
Automatically parsed Wikipedia	en_wiki-pred	T1, T2	-	-	1,313
	es_wiki-pred	T1, T2	-	-	1,280
	fr_wiki-pred	T1, T2	-	-	1,313
	ko_wiki-pred	T1	-	-	530
	pt_wiki-pred	T1	-	-	1,135
	ru_wiki-pred	T1	-	-	1,291

Table 6.1: Numbers of sentences of the datasets in SR'19 and SR'20, where the in-domain, out-of-domain, and automatically parsed treebanks are used in both shared tasks, and the automatically parsed Wikipedia treebanks are newly introduced in SR'20.

Tables 6.2 and 6.3 illustrate the data format for the T1 and T2. Comparing to the original UD format in Table 3.1, the order of the tokens in T1 and T2 is scrambled, and some additional information are introduced in the morphological features column to indicate the relative position of a token to its head (e.g.,`lin=+1` means the word follows its head). Comparing the T2 input to the T1 input, the representation is more abstract, since many function words are removed (e.g., determiner, punctuation, and preposition in this example). Some morphological features are underspecified (e.g., the person and number agreement in the verb *come*), while some others are transferred (e.g., the definiteness is carried from the removed determiner *the* to the noun *AP*). Finally, the dependency label are changed from the UD relations to the more semantics-oriented predicate-argument relations in the PropBank/NomBank style (Palmer et al., 2005; Meyers et al., 2004).

ID	Word	Lemma	UPOS	XPOS	Morphological Features	Head	Label	_	_
1	AP	_	PROPN	NNP	Number=Sing	5	obl	_	_
2	:	_	PUNCT	:	lin=+1	5	punct	_	_
3	this	_	DET	DT	Number=Sing|PronType=Dem	6	det	_	_
4	the	_	DET	DT	Definite=Def|PronType=Art	1	det	_	_
5	come	_	VERB	VBZ	Mood=Ind|Number=Sing|Person=3|...	0	root	_	_
6	story	_	NOUN	NN	Number=Sing	5	nsubj	_	_
7	from	_	ADP	IN	_	1	case	_	_

Table 6.2: An example sentence for the shallow track (T1).

ID	Word	Lemma	UPOS	XPOS	Morphological Features	Head	Label	_	_
1	AP	_	PROPN	_	Number=Sing|Definite=Def	2	AM	_	_
2	come	_	VERB	_	Tense=Pres|ClauseType=Par	0	ROOT	_	_
3	story	_	NOUN	_	Number=Sing	2	A1	_	_
4	this	_	DET	_	PronType=Dem	3	A1INV	_	_

Table 6.3: An example sentence for the deep track (T2).

6.2 Full Pipeline

We develop a pipeline system combining the linearization system described in Chapter 3 and several modules described in this section: (1) function word completion, (2) morphological inflection, (3) contraction, and (4) detokenization. Figure 6.1 illustrates a full pipeline for the shared task in both shallow and deep tracks. Linearization is the first step for both tracks, although the input structures are different. Additionally, in the deep track, we generate the function words after ordering the content words. Afterward, the

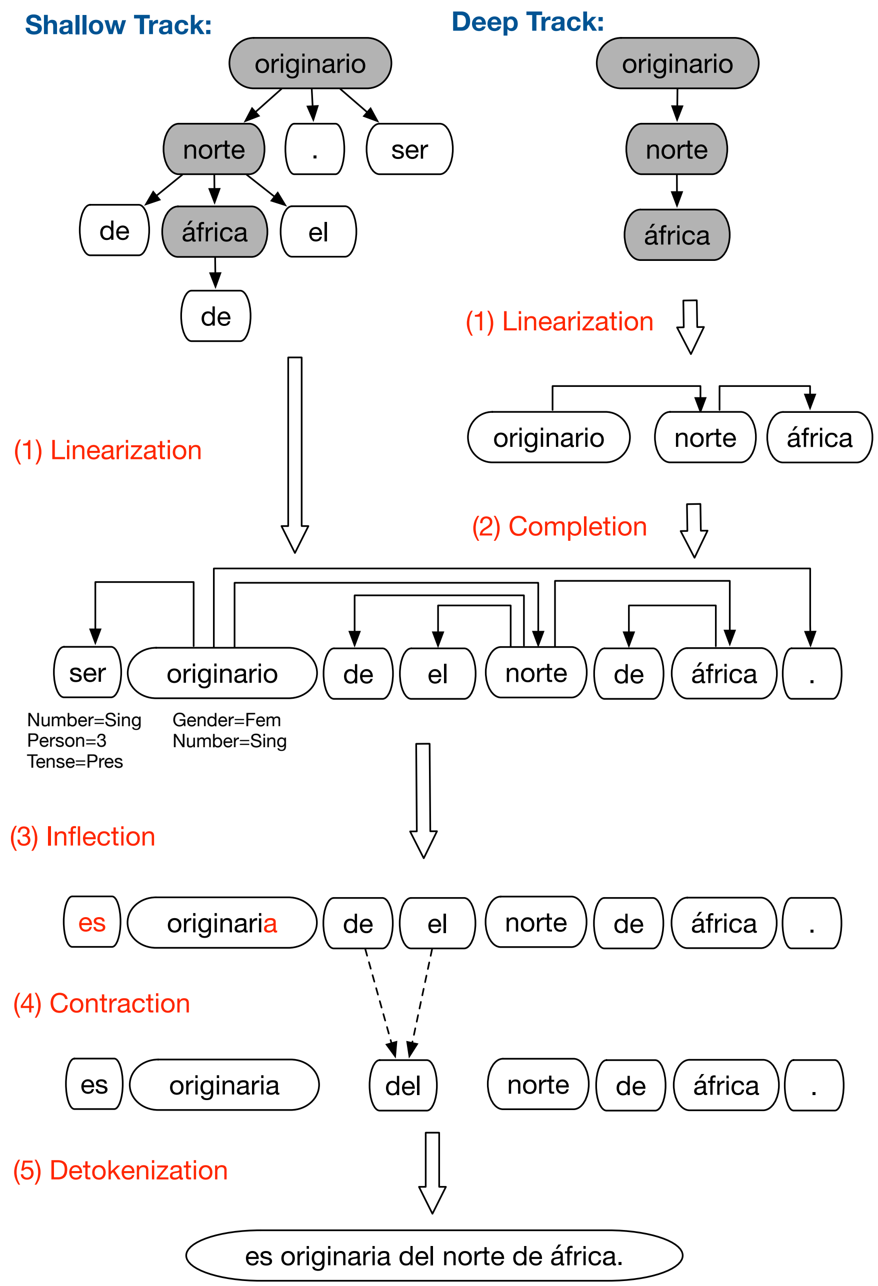


Figure 6.1: A full pipeline for the surface realization shared task, including five modules: (1) linearization, (2) completion, (3) inflection, (4) contraction, and (5) detokenization.

same steps apply for both tracks, i.e., generating the inflected word forms from the lemmata based on the morphological features and the context, contracting some words into one token, and finally detokenizing the tokens as a text string. Except for detokenizing, which is not backed by a machine learning model, all the other modules use the same encoder architecture with the same features, but the parameters are not shared, rather, each module is trained individually.

6.2.1 Linearization

Various options for the linearization module are extensively described in Chapter 3, which are gradually developed during the shared tasks.

As the encoder, we always use the bidirectional Tree-LSTM (§3.1.3) to represent the input dependency tree. It is more expressive than other alternatives as shown in §4.2. After encoding the input tree, we apply the divide-and-conquer strategy to order each subtree separately and then obtain the full projective order.

In the SR'19 shared task, we use the incremental beam search decoder (with a beam size of 32) to order the subtrees. For each subtree, we apply three decoders with different generation orders, left-to-right, right-to-left, and head-to-dependent, which share the same encoder. We use the majority vote to obtain the most likely order and use random selection for tie-breaking.

In the SR'20 shared task, we replace the incremental decoders with a TSP-based decoder (§3.3.2) and adding the transition-based reordering system (§3.3.3) to produce non-projective sentences.

6.2.2 Function Word Completion

In the deep track, the input tree consists of only the content words, constructed by a rule-based script from the organizers. In the UD annotation guideline, the function words are almost always attached to the content words and do not have dependents, which means the function words are always leaf nodes in the tree.

The completion module is responsible for generating the function words after the content words are linearized. It takes the sequence of content words in each subtree as input and inserts function words to complete the sequences. Like the head-first incremental linearization, it starts with each subtree's head word and attaches function words to the left and right sides of the head separately.

For each direction, we use a pointer to indicate the current token, which initially points to the head. At each step, we either predict a word from the vocabulary of function words and attach it next to the current token indicated by the pointer (and point to the newly generated token) or advance the pointer to the next token.

We use two pairs of LSTMs to encode the sequence: a pair of forward and backward LSTMs for the left dependents and a pair for the right dependents, where "forward" means from the head to the end and "backward" means from the end towards the head. Since the two pairs are symmetrical, we only describe the decoding process to the right side of the head.

We use the backward LSTM to encode the upcoming content words and the forward LSTM to encode the already processed words up to the pointer (including both the previously linearized content words and the newly generated function words).

At each decoding step, we concatenate the forward LSTM output of the current pointed token and the backward LSTM output of the next token and calculate a softmax distribution of all possible function words, as well as a special symbol $\Rightarrow$, which moves the pointer to the next token. If a new token is generated, the pointer will point to the new token. If $\Rightarrow$ is predicted and the pointer already reached the last token in the sequence, then the completion process is terminated.

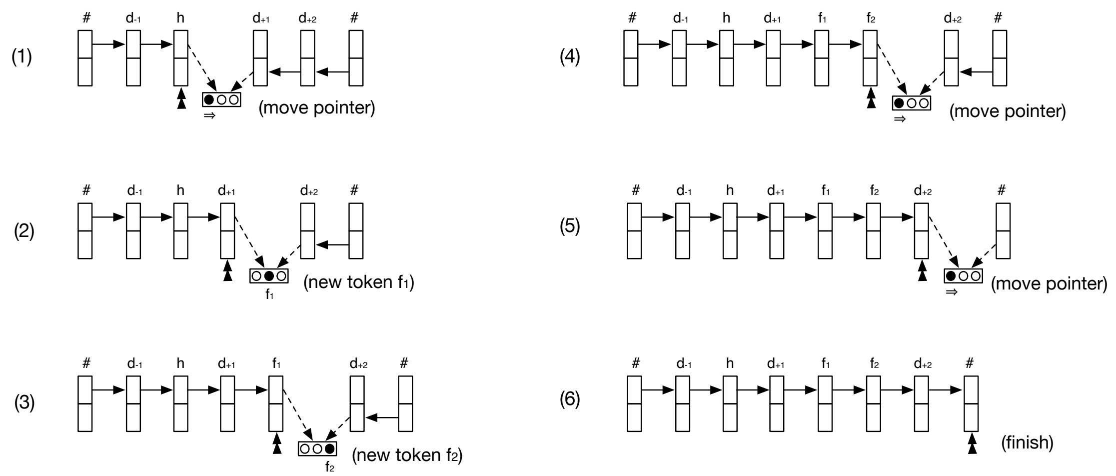


Figure 6.2: An example of the completion process to the right side, where the right arrows illustrate the forward LSTM, and the left arrows the backward LSTM.

Figure 6.2 illustrates an example of the completion process to the right side of the head, where the linearized content words are [#, $d^{(-1)}$, h, $d^{(+1)}$, $d^{(+2)}$, #], where h is the head, $d^{(-1)}$ is the left dependent, $d^{(+1)}$ and $d^{(+2)}$ are right dependents, and # is the boundary symbol.

We start with the pointer at the head and move it to the right step by step. The left dependents are encoded to provide contextual information for the generation to the right of the head. In step (1), the symbol $\Rightarrow$ is predicted, so we move the pointer from the h to

$d^{(+1)}$; in step (2), a new token f_1 is created and inserted next to $d^{(+1)}$; in step (3) another token f_2 is created and inserted next to f_1; in step (4) the pointer is moved to $d^{(+2)}$; in step (5) the pointer is moved again to #, which terminates the process and outputs the right half of the sequence [h, $d^{(+1)}$, f_1, f_2, $d^{(+2)}$]. A similar process is also applied to the left side of the head. Then, the outputs on both sides are combined to obtain the full sequence for the subtree.

The left and right completion processes are independent of each other since both forward LSTMs are only aware of the initial linearized tokens on both sides but not the newly generated tokens. We tried several variations in the preliminary experiments, including joint linearization and completion, interleaving the left and right completion processes, and beam-search for completion. All approaches yielded lower performance than the described method.

6.2.3 Morphological Inflection

After ordering all the tokens in the tree, the next step is to inflect each token's lemma into the word form according to the morphological features and the syntactic information. We use a hybrid approach for the inflection task. We first extract all inflection patterns from the training data: given the combination of the lemma, UPOS, and morphological features, if there is a word form appearing more than once and has over 99% certainty, then we keep it as a rule.

For the tokens not covered by the rules, we use a Seq2Seq model to predict the inflection similar to Kann and Schütze (2016), but with a major difference. Instead of the inflected word, we predict the edit script that modifies the lemma to the word as the output sequence. The alphabet of the edit script includes all the characters in the treebank and three special symbols: ✓ to copy one input character, ✗ to delete one input character, and $ to finish generation. For example, to inflect the Portuguese verb *falar* (to speak) to *falando* (speaking), the output sequence is "✓✓✓✓✗ndo$". The advantage of predicting edit scripts instead of words is that the copy action avoids the mistake of generating incorrect but similar characters.

The ground truth of such sequence is calculated from the Levenshtein edit operations between the lemma and the word form, where only insertion and deletion are allowed, i.e., no substitution.

Our model is in a way similar to the hard monotonic attention in Aharoni and Goldberg (2017), but we use a simpler source-target alignment, namely the Levenshtein edit operations based on dynamic programming, which does not require training, and thus the alignment quality is invariant to the amount of the training data. We use copying as an edit operation to avoid unnecessary generation errors while they directly generate the characters, even if the output character is identical to the aligned character in the input. Furthermore, our edit operations are associated with the pointer's movement, while they treat moving the pointer as an atomic operation, which leads to longer prediction sequences, thus more prone to error propagation.

We use a bidirectional LSTM to encode the characters in the lemma and the morphological tags as a sequence. Each character embedding is concatenated with a binary feature that indicates whether the corresponding input character is currently the target of the edit operation. Initially, the indicator for the first character is set to 1, and the rest are 0. A decoder LSTM with attention is then used to predict the edit operations. The input to the decoder LSTM state is the concatenation of the tree-structured token representation, the attended input vector, and the embedding of the last produced character. In the case of copy, we use the embedding of the copied character instead of the copy symbol ✓. If the predicted action is copy (✓) or delete (✗), the indicator on the input is then advanced by one step; if the prediction is adding a character, the indicator does not move.

6.2.4 Contraction

Contraction is a phenomenon in some languages in the shared task, such as Arabic, Spanish, French, and Portuguese, where several words are written as one token. The contracted word is sometimes but not always the simple concatenation of the component words. For example in Spanish, *hacer* (to do) and *lo* (it) are contracted into *hacerlo*, but *de* (of) and *el* (the) are contracted into *del*.

We implement a two-stage model to handle contraction. First, we treat it as a chunking problem and predict a BIO tag for each word in the sequence to indicate whether it is the beginning (B), inside (I), or outside (O) of a contraction group. For each contraction group, we then concatenate the words in it and feed to a Seq2Seq model to predict the contracted word form. An example is illustrated in Figure 6.3.

6.2.5 Detokenization

The final step is to render the sequence of tokens as a string of characters, known as detokenization. It mainly involves dealing with the white space between the words and punctuation. It can be effectively done with a rule-based off-the-shelf tool MosesDetok-

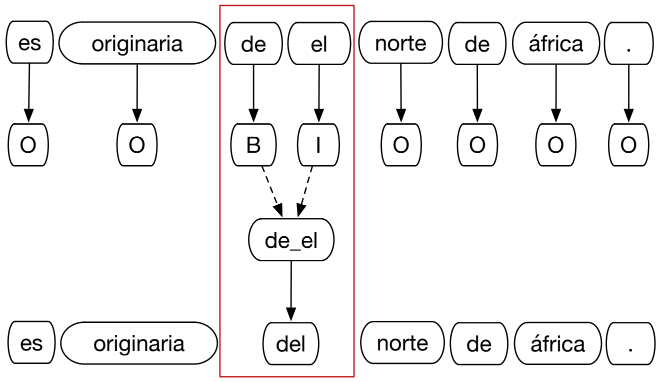


Figure 6.3: An illustration of the contraction model.

enizer.[21] The purpose of this step is to produce naturally written sentences for the human evaluation, while the automatic metrics are evaluated without detokenization so that the n-grams can be obtained to calculate the BLEU score.

6.3 Data Augmentation

As we have shown in §4.6, the linearization module is very data-efficient; it approaches high performance with only a few hundred sentences. SR'20 features an open subtrack, where unlimited unlabeled training is allowed. We thus try to push the performance with extra training data in this scenario.

For T1, we use the automatically annotated treebanks from the CoNLL 2017 shared task,[22] for T2, we use a subset of the English data from Elder et al. (2020) and for French and Spanish, the freely available news datasets from `webhose.io`.[23] All datasets are parsed with UDPipe (Straka and Straková, 2017) using the pre-trained models.

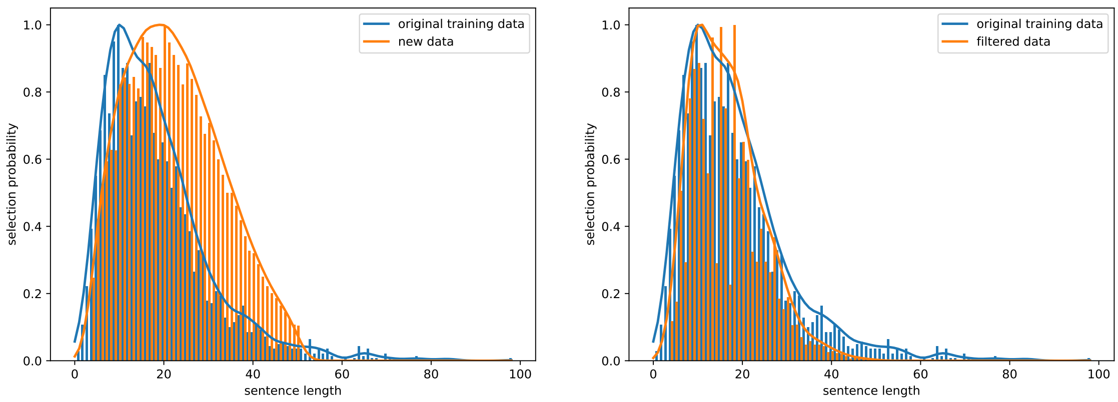
To maximize the structural similarity between the original data and the additional data, we filter the new datasets according to the original training data. We control for sentence length and branching factor by estimating their distributions in both the original training data and the new datasets. To efficiently filter using rejection sampling, we normalize the distributions in the original training data by the maximum value to obtain probabilities between 0 and 1.

During sampling, we decide whether we accept or reject a sentence according to the

[21] `https://pypi.org/project/mosestokenizer/`
[22] `https://lindat.mff.cuni.cz/repository/xmlui/handle/11234/1-1989`
[23] `https://webhose.io/free-datasets/language/`

product of both probabilities and the vocabulary overlap. The change of distribution after the filtering can be seen in Figure 6.4.

$$p(s) = D_1(s) \cdot D_2(s) \cdot \frac{|v_{original} \cap v_s|}{|v_s|} \tag{6.1}$$



(a) before filtering **(b)** after filtering

Figure 6.4: Sentence length distribution in en_lines before and after filtering.

We sample 200,000 sentences for each language and convert them into the shared task format as extra training data, which is only used to train the linearization and function word completion modules since it did not show significant improvement for other modules in the preliminary experiments. The reason could be that the accuracy of inflection and contraction is already quite high, and the augmented data could introduce more noise than signal. The largest improvement appears in the completion module since a vast amount of data is needed to properly learn semantics, especially for generating appropriate prepositions.

6.4 Ensemble

Ensembling has shown success in various NLP tasks, where several models are trained, and the prediction is made by majority voting among the models. We train ten instances of each module with the same hyperparameters but different random seeds and use majority voting to select the final output sentence. On the development sets, the ensemble prediction performs about 1 BLEU point higher than the single model on average.

For the open track, we train multiple models with different ratios between the original and the augmented data. Concretely, after training one step with the original data, we train N steps with the augmented data, where N ranges from 1 to 10. For different treebanks, the optimal value of N varies drastically. We simply take the majority prediction from these ten models to avoid individual tuning.

6.5 Shared Task Results

6.5.1 Evaluation Metrics

The results are evaluated with automatic metrics as well as by human judges. Among the automatic evaluation metrics, the BLEU score (Papineni et al., 2002) is the most commonly used, which calculates the precision of n-grams between the generated and the original text. The NIST score is based on BLEU but favors the n-grams that are more informative, i.e., more weight is given to the less frequent n-grams. The DIST score calculates the inverse normalized character-based string-edit distance.

In the human evaluation, a subset of the languages is selected and evaluated on Mechanical Turk,[24] where two quality criteria are assessed: **readability** (whether the generated sentence is natural and understandable) and **meaning similarity** (whether the generated sentence conveys the same meaning as the reference).

6.5.2 SR'19 Results

The automatic evaluation results of our submission to the shared task are shown in Tables 6.4 and 6.5 for the shallow and deep tracks, respectively. The first three columns contain the BLEU, DIST, and NIST scores of our system, and the fourth column is the difference of BLEU scores between our system and the best system among other participants for each treebank.

Our system achieves the best performance for all treebanks in both tracks. Compared to the best BLEU scores of other teams, our advantage ranges from single digits for the English treebanks to about 20 points for most other treebanks and 38 points for Arabic. In the out-of-domain scenario, our system performs very stably in most of the cases. However, compared to the English and Japanese PUD treebanks, the performance drop on the Russian PUD treebank is quite notable. Our conjecture is that the annotation guideline of the PUD treebank is much closer to the GSD treebank than the SynTagRus

[24] `https://www.mturk.com/`

treebank. Since we use both treebanks for training, the much larger size of SynTagRus might have dominated the training.

In the human evaluation (see Mille et al. (2019) for details), we also rank the first in all four evaluated languages (English, Russian, Chinese and Spanish) both for readability and adequacy.

	BLEU	NIST	DIST	Δ_{BLEU}
ar_padt	64.90	12.22	73.71	38.50
en_ewt	82.98	13.61	86.72	3.29
en_gum	83.84	12.69	83.49	1.45
en_lines	81.00	12.71	82.21	5.51
en_partut	87.25	11.01	85.68	8.27
es_ancora	83.70	14.69	79.82	7.23
es_gsd	82.98	12.77	79.45	12.83
fr_gsd	84.00	12.45	84.15	23.85
fr_partut	83.38	10.36	82.32	17.37
fr_sequoia	85.01	12.53	85.13	22.22
hi_hdtb	80.56	13.07	79.07	11.33
id_gsd	85.34	12.83	83.92	21.63
ja_gsd	87.69	12.42	87.17	24.10
ko_gsd	74.19	12.27	80.95	28.11
ko_kaist	73.93	13.00	78.69	26.70
pt_bosque	77.75	12.15	79.80	25.06
pt_gsd	75.93	13.07	79.33	23.43
ru_gsd	71.23	12.15	73.04	16.14
ru_syntagrus	76.95	15.08	78.66	16.96
zh_gsd	83.85	12.78	83.18	15.31
en_pud	86.61	13.47	87.00	2.54
ja_pud	86.64	13.02	84.04	20.12
ru_pud	58.38	10.91	77.12	6.01
en_ewt-pred	81.80	13.46	85.35	4.59
en_pud-pred	82.60	13.26	86.18	1.94
es_ancora-pred	83.31	14.61	81.14	6.03
hi_hdtb-pred	80.19	13.05	78.88	10.27
ko_kaist-pred	74.27	13.02	79.12	27.55
pt_bosque-pred	78.97	12.14	81.56	25.33
AVG	79.97	12.79	81.62	15.64

Table 6.4: Automatic Evaluation Results of the shallow track (T1) and the BLEU difference with the best system among other participants for each treebank.

	BLEU	NIST	DIST	Δ_{BLEU}
en_ewt	54.75	11.79	76.30	25.17
en_gum	52.45	11.04	73.07	25.85
en_lines	47.29	10.63	71.93	18.21
en_partut	45.89	9.03	67.45	17.04
es_ancora	53.13	12.38	68.58	16.15
es_gsd	51.17	10.82	68.85	16.52
fr_gsd	53.62	10.79	68.82	28.02
fr_partut	46.95	8.27	68.99	18.76
fr_sequoia	57.41	11.00	72.06	28.85
en_pud	51.01	11.45	72.31	24.45
en_ewt-pred	53.54	11.55	74.99	24.91
en_pud-pred	47.60	11.08	71.65	21.83
es_ancora-pred	53.54	12.36	70.02	16.13
AVG	51.41	10.94	71.16	21.68

Table 6.5: Automatic Evaluation Results of the deep track (T2) and the BLEU difference with the best system among other participants for each treebank.

6.5.3 SR'20 Results

In SR'20, we are the only team to submit predictions on all the tracks and languages. We thus mainly compare the results with our system from SR'19. The three automatic evaluation metrics (BLEU, NIST, and DIST) of the three settings (SR'19, SR'20 closed and open subtrack) for T1 and T2 are listed in Tables 6.7 and 6.8. Table 6.6 provides an overview of the performances with the average BLEU scores in both tracks for the three systems.

	2019	2020a	2020b
T1	80.17	83.76	84.25
T2	51.60	54.28	55.76
T1∩T2	84.24	86.53	86.88

Table 6.6: Comparing the average BLEU scores of our SR'19 results and our SR'20 results in the closed (2020a) and open (2020b) subtrack.

In the closed subtrack, our system improves over the previous year by 3.59 points in T1 and 2.68 points in T2. The improvement mainly comes from a better token representation, a better linearizer, and the model ensembling. In the open subtrack, the augmented data brings an additional gain of 0.49 in T1 and 1.48 in T2. When we only consider the a subset of the treebanks in T1 that also appear in T2 (T1∩T2), the gain of the additional data is only 0.35. It is clear that the additional training data are more beneficial for the T2

	BLEU			NIST			DIST		
	2019	2020a	2020b	2019	2020a	2020b	2019	2020a	2020b
ar_padt	64.90	69.56	69.71	12.22	12.63	12.62	73.71	75.80	76.51
en_ewt	82.98	86.16	85.67	13.61	13.78	13.74	86.72	88.48	87.74
en_gum	83.84	88.89	89.70	12.69	12.96	12.98	83.49	91.41	91.97
en_lines	81.00	85.05	85.30	12.71	12.98	12.97	82.21	85.89	86.48
en_partut	87.25	89.72	89.37	11.01	11.07	11.05	85.68	90.38	88.73
es_ancora	83.70	87.42	87.34	14.69	14.90	14.89	79.82	85.66	85.26
es_gsd	82.98	84.61	84.52	12.77	12.86	12.86	79.45	82.60	82.53
fr_gsd	84.00	86.08	85.08	12.45	12.58	12.55	84.15	84.64	83.24
fr_partut	83.38	87.09	89.22	10.36	10.56	10.58	82.32	85.84	87.67
fr_sequoia	85.01	87.25	87.29	12.53	12.65	12.64	85.13	85.65	85.12
hi_hdtb	80.56	84.53	84.77	13.07	13.32	13.34	79.07	83.03	83.14
id_gsd	85.34	87.53	88.33	12.83	12.89	12.91	83.92	86.41	87.11
ja_gsd	87.69	89.36	89.54	12.42	12.53	12.54	87.17	87.00	87.83
ko_gsd	74.19	81.14	82.52	12.27	12.44	12.46	80.95	85.49	86.82
ko_kaist	73.93	79.96	80.28	13.00	13.18	13.19	78.69	84.52	84.90
pt_bosque	77.75	82.92	83.36	12.15	12.40	12.41	79.80	84.59	85.45
pt_gsd	75.93	80.59	80.69	13.07	13.33	13.34	79.33	87.86	87.70
ru_gsd	71.23	77.43	78.93	12.15	12.40	12.42	73.04	78.89	81.54
ru_syntagrus	76.95	81.82	79.78	15.08	15.45	15.34	78.66	83.02	81.07
zh_gsd	83.85	86.36	88.05	12.78	12.86	12.91	83.18	83.89	85.19
en_pud	86.61	85.37	85.65	13.47	13.41	13.41	87.00	85.75	84.52
ja_pud	86.64	88.88	89.21	13.02	13.20	13.23	84.04	85.67	86.29
ru_pud	58.38	69.07	69.35	10.91	11.85	11.89	77.12	82.40	82.67
en_ewt-pred	81.80	84.72	84.31	13.46	13.61	13.59	85.35	86.65	86.24
en_pud-pred	82.60	79.74	80.14	13.26	13.16	13.16	86.18	82.19	82.23
es_ancora-pred	83.31	86.97	86.81	14.61	14.81	14.80	81.14	86.96	86.36
hi_hdtb-pred	80.19	84.42	84.78	13.05	13.31	13.33	78.88	83.52	84.05
ko_kaist-pred	74.27	81.01	81.42	13.02	13.22	13.23	79.12	85.36	85.82
pt_bosque-pred	78.97	84.35	85.18	12.14	12.40	12.43	81.56	87.35	87.98
en_wiki-pred	86.54	88.34	90.85	13.91	14.00	14.14	86.52	86.51	89.21
es_wiki-pred	84.97	87.85	88.28	13.95	14.09	14.10	81.39	87.35	87.68
fr_wiki-pred	87.79	89.22	90.60	14.00	13.98	14.09	86.44	91.64	92.58
ko_wiki-pred	73.81	76.57	81.65	11.37	11.50	11.55	79.87	80.20	85.99
pt_wiki-pred	79.69	83.57	84.74	13.25	13.57	13.61	81.54	84.58	86.40
ru_wiki-pred	73.86	77.91	76.31	12.65	12.86	12.86	77.26	82.37	80.09
AVG	80.17	83.76	84.25	12.86	13.05	13.06	81.71	85.13	85.55

Table 6.7: Automatic evaluation results in the shallow track (T1) of the our system in SR'19 and the closed subtrack (2020a) and open subtrack (2020b) in SR'20.

track since the function word completion module requires more training data.

In the human evaluation on three languages, our systems (with or without data augmentation) come in second place for English in terms of readability and meaning similarity and rank first in both aspects for Russian and Spanish. A detailed report is in Mille et al. (2020).

	BLEU			NIST			DIST		
	2019	2020a	2020b	2019	2020a	2020b	2019	2020a	2020b
en_ewt	54.75	58.84	58.66	11.79	12.13	12.14	76.30	78.99	79.23
en_gum	52.45	53.98	56.33	11.04	11.09	11.25	73.07	73.22	76.47
en_lines	47.29	50.23	50.45	10.63	10.82	10.89	71.93	73.06	73.10
en_partut	45.89	46.87	50.11	9.03	8.92	9.26	67.45	69.75	72.98
es_ancora	53.13	56.67	55.64	12.38	12.66	12.66	68.58	71.85	71.52
es_gsd	51.17	55.10	55.99	10.82	11.05	11.13	68.85	71.01	72.53
fr_gsd	53.62	58.86	56.95	10.79	11.14	11.06	68.82	72.38	71.94
fr_partut	46.95	51.11	57.62	8.27	8.38	8.88	68.99	68.43	75.04
fr_sequoia	57.41	59.37	60.26	11.00	11.02	11.21	72.06	73.71	73.30
en_pud	51.01	50.59	53.70	11.45	11.36	11.59	72.31	70.43	73.31
en_ewt-pred	53.54	57.10	57.14	11.55	11.88	11.92	74.99	77.23	76.98
en_pud-pred	47.60	47.96	50.15	11.08	11.02	11.19	71.65	70.53	72.76
es_ancora-pred	53.54	56.92	55.96	12.36	12.63	12.63	70.02	73.27	73.06
en_wiki-pred	49.74	52.80	56.26	11.56	11.82	12.12	71.46	72.53	74.94
es_wiki-pred	51.69	52.86	56.19	11.22	11.72	12.04	71.46	73.50	75.84
fr_wiki-pred	55.74	59.22	60.68	11.72	12.06	12.28	72.53	73.68	76.64
AVG	51.60	54.28	55.76	11.04	11.23	11.39	71.28	72.72	74.35

Table 6.8: Automatic evaluation results in the deep track (T2) of the our system in SR'19 and the closed subtrack (2020a) and open subtrack (2020b) in SR'20.

6.6 Summary

In this chapter, we put our linearization model into a pipeline system, together with the function word completion, morphological inflection, and contraction modules, to tackle the surface realization shared tasks. Our system achieved the overall best performance in SR'19 by a large margin.

To further push the performance, we experiment with data augmentation methods in SR'20 by parsing extra texts as additional training data. We focused on filtering and sampling the data so that they have a similar distribution of structural properties to the original training data. The augmented data mainly improves the performance in the deep track, particularly the function word completion module. In contrast, the improvement for the linearization module is much milder. The reason for the different degrees of improvement is that the function word completion is the most hungry for semantic knowledge from a vast amount of extra data, while the original training data size is quite sufficient for learning the word order.

Overall, the results demonstrate that our surface realization system is effective and stable across various languages. The system is relatively light-weight and requires very little data to achieve strong performances. In particular, linearization performance is close to the upper-bound (as shown in the human evaluation on the shallow track), while there is still ample room for improvement in the deep track, which is a more realistic scenario for practical NLP applications.

Chapter 7

Conclusion

The dependency grammar models the dependency relation between words, which renders a tree structure as the representation of a sentence. By design, the dependency grammar disregards the linear order and only focuses on the structural order, which contributes to its simplicity and universality. However, when we generate a natural sentence from a syntactic or semantic dependency tree, we need to determine its linear word order, which is the research focus of this dissertation.

7.1 Contributions

Developing a stable and efficient linearization model. The majority of our work focuses on developing a computational model to linearize a dependency tree, as described in Chapter 3. The linearization model consists of two main components: (1) an encoder that represents the structural order of the dependency tree and (2) a decoder that produces the linear order of the sentence.

We develop several encoders with different structural properties and decoders from different paradigms for structured prediction. We then explore the optimal combination in the experiments in Chapter 4, which conclude that the Tree-LSTM model can best encode the dependency tree compared to the sequential LSTM or GNN models. It can efficiently encode the structural information of the tree into each node.

To decode the linear order, it is best to apply the divide-and-conquer strategy and decode each subtree separately. It has two main advantages: (1) the search space of each subtree is much smaller thus allowing a simpler scoring model and decoding algorithm, and (2) the inductive bias of only producing projective trees benefits from the heavily skewed distribution toward projectivity in natural languages. The decoding algorithm treats the word ordering problem as a Traveling Salesman Problem, where each word is a city and the score of each word bigram is the traveling cost from one city to another. The bigram score is calculated with a biaffine attention model adapted from the work for dependency parsing.

Through the experiments we can conclude that the resulting linearization model has very a stable performance to process a typologically diverse set of languages. It is also shown to be able to learn with very little training data.

In Chapter 6, we apply the linearization model in a pipeline for the task of surface realization, which aims at generating sentences from a deep syntactic representation. The pipeline includes more subtasks such as function word generation, morphological inflection, and contraction. According to both automatic and human evaluations in two recent editions of the Multilingual Surface Realization shared tasks (SR'19, SR'20), our system achieved the best performance in almost all evaluated languages.

Impact of dependency length minimization on word order. In §4.7, we analyze the cases where the model prediction deviates from the observed order in the actual sentences and finds out that many are acceptable alternative orders. In fact, since a dependency tree's correct surface order is not necessarily unique, the frequency of errors (especially from a model with high performance) actually reflects the freedom of word order in a language. This leads our focus to a quantitative linguistic analysis on the freedom of word order, particularly the factor that influences word order, as laid out in Chapter 5.

Dependency Length Minimization (DLM) is an influential hypothesis that describes a preference to arrange the word order in a way that reduces the total length of the dependency arcs (DL) in a sentence. Several studies have demonstrated the effect of DLM in a large-scale collection of languages by showing the actual sentences generally have shorter DL than random linearizations, providing evidence for the validity of the hypothesis. We complement the previous work by breaking down the word order constraints into more fine-grained aspects, including the distribution of the dependents, the direction of each individual dependent, and the arrangement of siblings. By constructing baselines that reflect each aspect and comparing the dependency lengths, we show that DLM indeed manifests itself in all these aspects.

Next, we show that DLM can explain the emergence of the word order preference: if we flip the pair of words in the actual data that follows the preferred order in the language, the DL would be longer. More interestingly, DLM can also explain the exception in the word order preference: if we flip the pair of words that goes against the preferred order, the DL would also be longer.

Treatment of non-projective sentences. Most languages have a strong bias toward projectivity. That is, the words tend to be arranged in a way that the dependency arcs do not cross each other. With the assumption of projectivity, we can apply the divide-and-conquer strategy to linearize the subtrees separately, which significantly reduces the search space. This simplification rules out some legitimate non-projective sentences. To remedy this problem, we introduce a transition system to minimally reorder a projective sentence into a non-projective sentence. Experiments show that this treatment can recover a considerable portion of non-projective sentences while has a minimal negative impact on the projective sentences.

In the study on DLM, we also apply the experimental setting to explore the relation of DLM and (non-)projectivity, another aspect of word order often overlooked in the previous work. The result suggests that projectivity is similarly influenced by DLM. The majority of sentences are projective since a random projective ordering would much shorter DL than a random non-projective one; therefore, non-projective sentences are rare in natural language. However, when non-projective sentences do occur, they also have shorter DL than their projective version with the same local order, which means the violation of projectivity could also be motivated by DLM.

7.2 Future Perspectives

Generation from more general representations In this thesis, we focus on linearizing a specific type of input structure, the dependency tree, which has many properties that facilitates processing. First, the shallow dependency tree is a bilexical representation, which means every word in the sentence appears exactly once in the structure. This means that the linearization algorithm does not need to decide if it should remove or add any words. As we have seen in the case of the deep dependency tree, adding function words is much more difficult than just ordering the words. Our model also greatly benefits from the tree property of the dependency grammar. It allows us to use the Tree-LSTM to encode the structure efficiently. The divide-and-conquer strategy also can only work on trees but not general graphs.

Many other syntactic or semantic formalisms are not restricted by the bilexical and tree properties, which could be more expressive but less convenient for the model to process (both parsing and linearization). For example, several semantic dependency graph representations in Oepen et al. (2014) are based on the dependency tree but augmented with extra edges that break the tree property. Another sentence representation widely adopted in NLP is the Abstract Meaning Representation (AMR) (Banarescu et al., 2013). The AMR represents a sentence with a rooted, directed, (mostly) acyclic graph. A node represents a concept instead of a word, and an edge represents a semantic relation between two concepts. Data-to-text generation tasks such as the E2E NLG Challenge (Dušek et al., 2018) and the WebNLG Challenge (Gardent et al., 2017) aims at generating texts from knowledge graphs, which are also free from the bilexical and tree properties.

Thus, it is a challenge how to generate sentences based on these representations while getting the most out of the experience on linearizing dependency trees. We believe that the graph-based scoring model and decoding algorithm could still be of tremendous help.

Syntax-based Machine Translation Linearization and parsing are a pair of tasks that goes in the opposite direction: linearization transforms the structural order to the linear order, parsing transforms the linear order to the structural order. Although this thesis mainly focuses on linearization, there is a lot of potentials to combine both directions.

The most promising perspective is syntax-based machine translation, especially between language pairs with little training data or significant word order differences. Recent work on machine translation has been dominated by end-to-end models trained on an enormous amount of data, primarily variants of the Transformer model (Vaswani et al., 2017). The end-to-end models do not explicitly model the syntax or word order; instead, it assumes that the deep learning model can implicitly learn to understand the source sentence's syntactic structure without parsing and producing the target sentence in an auto-regressive fashion. This type of model has achieved impressive translation quality. However, it heavily relies on the availability of large parallel data, and it does not work well for low-resource languages.

Our linearization model is very data-efficient. As shown in the experiments, it requires only a few hundred training instances to achieve reasonable performance. A typical dependency parsing model also needs a similar amount of data to be of practical use. In addition to these two components, a lexical translation model is needed to map the nodes in the syntax tree of the source language to the target language. With proper annotated resources, we also could model deeper structure beyond the surface-syntactic structures, and perform the transfer on the deep-syntactic structures, as envisioned in Mel'cuk et al. (1988). Generally, we are optimistic about the efficiency and performance of the syntax-based translation model for low-resource languages.

Appendix A

Detailed Experimental Results

	TSP-sub						TSP-full					
	bag	seq	seq-nb	seq-rd	tree	graph	bag	seq	seq-nb	seq-rd	tree	graph
af	82.18	82.67	80.62	80.72	84.53	85.55	22.88	77.50	43.57	20.58	81.18	73.12
ar	84.35	85.48	84.51	84.14	85.75	85.56	32.93	82.04	53.48	32.48	80.85	73.59
bg	83.26	84.44	82.92	83.12	84.65	84.53	41.33	80.44	61.71	40.31	80.94	76.73
ca	85.94	87.13	86.02	86.18	87.71	87.29	35.05	83.31	58.32	33.38	81.90	77.52
cs	72.91	75.47	72.60	72.63	72.19	75.70	34.35	71.56	52.65	31.59	72.77	66.12
cu	63.34	62.72	62.44	62.08	63.53	62.50	39.92	57.68	45.87	38.26	59.48	54.78
da	76.57	78.07	75.39	76.18	78.78	79.38	33.15	74.91	50.31	30.83	76.58	70.46
de	74.69	76.38	74.99	74.44	77.12	76.54	42.39	72.86	58.21	40.10	74.47	71.26
el	82.26	83.26	81.35	81.94	83.54	83.16	37.44	76.04	51.60	38.14	77.22	74.00
en	84.34	85.60	84.06	83.63	85.58	85.87	41.85	82.62	61.84	40.53	82.42	79.27
es	83.96	84.87	84.34	83.76	85.04	84.75	31.37	81.99	56.68	29.09	80.04	76.34
et	67.21	69.27	67.62	67.27	70.41	70.36	28.77	65.65	48.48	27.00	65.30	60.79
eu	59.34	60.02	58.30	58.00	60.62	60.64	35.17	60.70	47.42	34.14	61.63	58.35
fa	82.67	84.51	82.21	82.82	84.84	85.07	25.66	81.33	50.58	25.26	81.24	73.78
fi	76.18	78.14	76.16	75.44	77.95	77.75	41.06	76.64	64.80	40.31	76.56	73.01
fr	89.18	89.67	89.50	88.93	90.20	90.11	42.64	87.85	67.72	39.73	86.50	83.61
fro	68.39	67.71	67.18	67.89	68.31	67.53	41.20	67.62	56.33	39.22	67.27	65.19
gl	80.83	82.17	80.65	80.55	82.42	82.56	22.66	77.17	47.61	22.42	76.72	71.57
got	58.33	58.87	57.65	57.81	58.99	59.94	32.24	54.42	39.95	31.63	54.29	51.18
grc	54.77	56.19	54.59	55.30	56.93	56.93	31.95	53.66	40.63	30.18	53.38	49.45
he	83.53	83.43	82.93	82.47	84.25	84.27	32.44	79.45	53.60	31.12	79.05	75.69
hi	83.98	84.17	83.83	83.55	84.13	83.99	52.10	84.18	70.39	51.81	83.87	80.91
hr	77.29	79.05	77.26	76.80	80.83	79.93	31.49	74.19	48.29	29.19	74.81	68.90
hu	61.56	63.23	60.41	60.95	64.61	63.50	16.26	53.96	32.66	15.83	57.04	51.21
hy	63.97	66.29	64.64	64.15	66.51	68.76	19.85	51.05	30.30	18.37	59.11	50.41
id	79.77	83.23	79.38	78.52	83.83	83.50	29.40	77.54	52.12	28.00	76.97	71.59
it	84.04	84.70	83.25	84.00	85.08	85.28	40.66	81.39	59.06	35.49	80.78	77.91
ja	90.82	90.55	90.91	90.50	91.08	91.15	42.98	88.76	67.53	40.81	86.59	85.07
ko	76.92	79.51	78.68	76.04	80.03	79.86	36.12	81.05	69.11	34.50	81.49	76.92
la	59.39	59.64	59.00	58.52	59.83	59.85	33.72	58.98	46.50	32.85	58.23	53.37
lt	66.30	69.64	66.05	64.47	69.59	70.29	17.78	51.85	30.03	17.51	58.05	51.88
lv	71.28	74.01	71.68	72.17	75.01	74.17	27.33	68.15	46.59	25.88	68.38	64.12
lzh	87.03	87.16	87.02	86.99	87.84	88.22	62.72	86.03	76.72	61.09	86.97	84.48
mt	78.84	79.52	78.60	79.51	80.36	80.40	19.83	69.30	36.05	19.40	71.81	65.76
nl	75.07	76.02	74.18	73.95	75.77	75.72	35.21	73.98	54.31	32.73	73.51	70.14
no	84.16	85.70	83.97	83.71	85.92	85.86	40.21	83.33	63.60	38.40	82.65	78.23
orv	58.56	58.71	57.66	58.39	59.89	59.77	39.78	56.66	46.34	38.69	55.94	53.21
pl	76.12	77.01	75.24	75.60	77.27	77.53	36.41	73.24	54.32	34.70	73.01	67.82
pt	87.25	88.23	87.53	87.10	88.95	88.62	36.89	84.95	61.78	35.12	83.82	80.46
ro	76.60	77.65	76.53	76.56	78.34	78.20	40.10	76.36	59.12	38.81	75.06	72.57
ru	77.11	79.46	77.73	77.29	79.62	79.68	34.25	75.20	55.20	32.63	76.32	71.16
sk	72.82	73.20	72.91	72.27	74.03	74.74	32.53	66.53	48.27	30.44	68.45	63.79
sl	73.45	75.26	73.45	73.78	76.33	76.14	33.06	71.27	47.29	29.50	71.62	67.27
sr	80.67	82.44	79.67	79.33	83.11	82.66	35.11	77.98	49.59	33.64	78.83	73.25
sv	78.20	79.01	78.06	77.61	80.30	79.55	28.19	72.49	46.95	26.92	73.74	66.90
te	87.57	85.18	87.09	85.43	89.08	85.91	79.49	80.73	79.92	74.51	84.74	82.15
tr	67.35	70.80	68.57	66.45	72.15	72.39	32.25	65.98	43.50	31.85	66.02	59.62
ug	74.96	77.84	75.68	73.05	78.83	79.33	32.84	72.84	46.42	31.59	72.87	69.46
uk	76.21	77.57	76.23	75.78	78.53	78.62	27.61	70.33	44.42	26.83	69.92	65.36
ur	79.87	80.73	80.18	80.30	80.62	81.27	42.61	78.83	57.74	39.87	78.33	73.93
vi	74.42	77.30	73.95	73.86	78.25	78.09	23.38	69.97	38.59	22.96	71.88	63.42
wo	81.31	81.42	80.50	80.06	83.66	83.75	29.93	73.09	43.63	29.28	74.27	69.23
zh	81.62	82.28	80.82	81.67	83.27	83.55	27.81	76.25	46.75	26.14	76.04	69.72
AVG	76.20	77.41	75.98	75.73	78.04	78.04	34.80	73.05	52.16	33.24	73.60	68.98

Table A.1: Detailed BLEU scores for the experiment on encoders in §4.2.

	TSP-sub						TSP-full					
	bag	seq	seq-nb	seq-rd	tree	graph	bag	seq	seq-nb	seq-rd	tree	graph
af	24.23	23.71	22.68	23.20	32.47	35.05	2.06	23.71	6.19	2.06	28.87	20.62
ar	31.57	30.47	28.93	28.71	33.44	32.23	6.49	29.70	10.23	6.60	28.16	21.67
bg	49.33	49.33	47.00	47.71	51.57	51.84	20.63	47.89	34.08	19.91	49.96	45.74
ca	32.18	33.18	31.25	30.84	37.10	34.58	3.80	27.91	11.06	3.98	27.74	22.47
cs	30.67	33.80	29.68	29.09	29.75	34.71	16.60	32.28	23.03	15.50	33.09	28.12
cu	41.38	41.38	41.19	40.91	40.07	39.79	34.11	41.66	36.25	32.43	40.07	39.14
da	40.25	40.78	37.23	37.59	40.96	43.26	17.38	39.18	23.40	17.20	42.20	36.35
de	35.29	37.92	35.67	35.17	39.55	38.30	15.77	33.92	24.41	14.52	38.05	33.92
el	30.52	29.03	26.80	28.29	28.29	29.78	8.19	24.81	10.92	8.93	21.59	21.84
en	63.99	66.08	63.14	63.29	65.83	66.53	40.51	64.34	50.85	39.21	63.39	61.74
es	26.42	26.78	25.82	24.79	29.81	28.23	5.50	24.30	9.61	4.90	22.01	20.19
et	29.57	31.33	29.50	29.06	32.96	32.67	16.35	30.34	22.59	14.75	30.18	28.58
eu	19.47	20.80	19.35	18.91	20.08	21.58	12.96	21.86	16.74	12.57	23.53	23.08
fa	33.56	38.90	32.72	32.89	40.23	39.07	8.68	36.23	13.69	8.68	34.72	28.38
fi	59.47	62.29	60.16	58.77	61.28	61.65	36.37	61.97	53.33	36.11	63.15	60.00
fr	50.34	51.56	51.42	48.37	53.39	54.13	9.69	48.98	25.61	8.40	46.07	42.34
fro	42.02	43.27	41.69	41.15	42.45	42.29	33.39	44.63	41.26	32.90	44.84	45.28
gl	15.35	18.14	14.07	13.72	20.00	19.30	0.35	13.14	0.35	0.35	11.51	7.21
got	31.47	32.89	31.88	32.28	33.40	32.49	24.87	33.40	27.31	24.97	33.81	31.88
grc	24.04	22.67	21.79	21.10	24.14	22.87	17.66	25.61	21.30	17.37	26.79	22.77
he	33.68	33.06	33.88	31.20	36.57	37.19	8.26	31.40	13.22	8.88	28.51	26.24
hi	40.57	40.33	40.14	39.06	41.29	40.57	11.93	42.19	26.22	11.57	42.56	36.71
hr	21.25	25.00	19.79	19.79	28.54	27.08	5.10	20.94	7.50	4.38	22.40	17.40
hu	5.44	7.03	5.44	6.12	9.98	7.71	1.13	4.08	2.27	0.68	4.08	5.44
hy	22.47	24.39	22.65	20.03	22.65	25.26	10.63	18.47	12.54	10.98	22.82	17.77
id	37.75	44.19	35.60	33.99	45.80	43.65	9.12	39.18	20.75	8.05	38.64	35.42
it	42.20	42.91	41.67	42.38	46.28	46.10	16.67	40.78	24.65	13.65	40.60	38.48
ja	57.93	55.38	56.95	56.36	57.53	58.90	12.92	54.21	34.05	11.35	51.27	48.53
ko	42.21	47.43	45.84	41.09	48.16	48.06	16.26	51.89	40.03	15.20	53.10	48.60
la	24.75	24.18	23.61	23.13	24.46	24.56	18.90	27.18	22.23	18.47	24.94	24.04
lt	23.79	26.70	22.17	19.58	26.21	24.60	8.58	16.34	8.90	8.41	21.68	18.61
lv	31.57	32.80	30.83	30.83	35.57	34.59	15.91	31.01	22.69	16.21	33.85	29.41
lzh	89.20	89.40	89.35	89.35	89.99	90.91	69.27	88.47	82.17	68.78	89.74	87.84
mt	25.87	27.02	26.33	26.33	27.25	27.48	6.00	21.25	7.39	6.24	21.25	18.24
nl	33.98	36.49	33.15	31.34	35.10	34.40	13.79	34.82	20.47	12.12	34.82	30.64
no	56.29	58.32	55.21	54.96	59.73	58.78	27.23	56.58	41.88	25.20	55.13	51.22
orv	39.90	40.06	38.28	39.52	41.52	40.82	33.53	40.39	36.45	32.61	40.17	39.90
pl	35.52	37.37	35.39	33.36	36.92	37.01	15.40	35.07	24.45	14.63	35.25	31.07
pt	39.50	41.49	40.08	38.43	45.21	44.55	7.02	36.45	18.26	5.79	36.28	32.48
ro	34.79	37.45	35.74	35.84	38.50	39.07	17.11	37.55	25.10	17.68	37.74	34.22
ru	34.71	37.77	35.24	34.05	38.35	38.40	14.40	34.64	22.86	13.78	36.59	32.06
sk	43.87	43.40	45.28	43.02	45.09	45.57	25.00	42.26	32.17	23.87	42.08	38.40
sl	23.84	24.80	22.07	22.07	30.11	29.56	8.72	25.07	12.53	7.36	24.25	22.21
sr	27.61	32.65	26.49	25.00	32.84	33.58	7.09	26.31	9.14	5.60	25.56	23.13
sv	34.52	35.52	33.53	33.73	39.09	37.30	12.30	33.33	19.84	9.52	32.94	28.37
te	84.73	82.44	82.44	83.21	86.26	81.68	78.63	77.86	77.86	70.99	83.21	80.92
tr	44.64	47.77	45.04	42.81	49.49	49.29	30.87	48.08	35.93	30.67	46.86	44.94
ug	42.67	47.56	44.44	39.44	49.44	50.67	19.00	44.67	25.78	18.11	45.89	44.89
uk	31.85	32.89	31.25	30.65	37.20	34.38	11.46	27.98	15.48	11.01	28.27	25.89
ur	26.45	28.62	26.63	28.26	27.54	29.53	7.43	29.71	12.50	6.16	27.36	22.64
vi	34.75	40.38	35.00	34.25	42.50	41.88	9.62	33.50	12.62	8.62	37.00	28.88
wo	34.30	35.63	32.74	31.40	38.98	38.31	6.46	30.73	11.80	6.24	29.40	25.61
zh	27.40	26.20	25.20	25.80	28.20	28.40	3.60	22.00	7.20	4.60	19.80	18.60
AVG	36.62	38.13	36.03	35.32	39.61	39.44	16.81	36.04	23.57	16.01	36.30	33.02

Table A.2: Detailed EMF scores for the experiment on encoders in §4.2.

	TSP-sub						TSP-full					
	bag	seq	seq-nb	seq-rd	tree	graph	bag	seq	seq-nb	seq-rd	tree	graph
af	89.17	89.80	88.73	88.38	91.13	91.42	35.49	84.07	55.10	32.35	86.52	78.77
ar	91.30	91.71	90.94	90.83	92.08	92.00	50.67	89.57	69.52	50.38	88.50	83.30
bg	87.59	87.72	86.42	86.87	88.53	88.69	52.09	84.62	70.63	51.17	84.88	81.34
ca	89.17	89.37	88.46	88.33	90.66	90.03	41.99	85.87	66.13	39.86	84.88	80.97
cs	82.69	84.20	81.91	81.91	82.32	84.75	52.61	80.77	67.45	51.29	81.21	76.45
cu	74.80	74.72	74.12	74.07	74.92	74.32	56.21	70.38	62.29	55.58	71.59	68.44
da	85.62	86.81	84.87	85.20	86.92	87.95	44.39	80.96	61.65	44.22	81.77	76.89
de	82.75	84.02	83.05	82.59	84.59	83.97	57.77	80.63	69.69	56.95	81.37	79.23
el	87.05	86.95	85.43	85.90	87.50	87.34	45.78	81.41	61.13	46.23	81.26	78.13
en	88.50	89.51	87.99	88.01	89.60	89.72	49.88	87.05	70.52	48.98	86.09	83.48
es	87.17	87.47	86.75	86.40	88.46	87.80	39.26	84.64	64.70	37.68	83.17	79.77
et	77.78	79.16	78.02	77.62	80.18	80.07	48.35	76.72	64.90	47.23	75.94	72.76
eu	76.96	77.68	76.26	75.89	78.26	78.38	55.13	74.45	65.15	54.46	74.71	72.44
fa	90.69	92.26	90.50	90.48	92.38	92.42	45.82	89.64	68.22	45.54	89.84	83.92
fi	86.17	87.84	86.43	85.75	87.42	87.21	60.33	85.57	78.32	59.86	85.55	83.34
fr	91.13	91.31	91.08	90.56	91.93	91.72	44.16	89.33	72.29	41.91	88.27	85.37
fro	79.60	79.57	79.30	79.14	79.95	79.47	55.68	77.82	70.00	54.95	77.10	75.60
gl	84.59	85.47	83.96	83.85	86.01	85.87	26.42	81.36	55.26	25.89	80.38	75.04
got	71.79	72.68	71.62	71.91	72.85	73.27	50.98	68.42	58.21	51.35	68.99	65.27
grc	70.67	71.55	70.45	70.60	72.32	72.08	50.87	67.73	58.79	49.74	67.77	64.22
he	89.98	89.87	89.19	89.04	90.47	90.55	46.04	86.64	68.94	44.28	86.40	83.53
hi	90.71	90.93	90.62	90.47	91.04	90.79	59.76	89.78	78.62	59.27	89.09	86.62
hr	84.63	85.40	84.12	83.93	87.10	86.45	48.93	82.03	63.82	48.42	81.91	76.96
hu	78.57	80.00	77.72	78.31	81.18	80.79	39.86	71.91	56.40	38.38	73.73	68.67
hy	76.29	78.41	76.19	75.92	78.27	80.12	39.57	67.95	50.69	39.12	72.49	65.81
id	86.04	88.92	85.54	85.09	89.36	89.20	45.71	84.24	64.97	43.26	82.56	78.20
it	87.41	87.79	86.82	87.27	88.33	88.40	44.89	85.21	65.51	39.57	84.53	81.31
ja	93.05	92.93	92.90	92.61	93.25	93.43	45.75	91.39	73.36	45.89	89.51	87.74
ko	90.08	92.07	91.54	89.61	92.38	92.24	63.17	91.42	85.11	61.85	91.25	88.91
la	71.74	71.60	71.60	71.16	71.66	71.99	48.98	69.36	60.17	48.14	68.42	63.87
lt	78.68	80.72	78.03	77.30	80.90	81.22	41.46	67.85	51.53	40.85	72.14	66.59
lv	79.31	80.79	79.11	79.11	82.02	81.51	43.70	76.63	60.42	43.41	75.98	72.75
lzh	94.73	94.73	94.82	94.73	95.16	95.68	79.25	94.11	89.45	78.72	94.89	93.08
mt	85.30	85.25	84.95	85.17	85.90	85.97	38.75	78.40	52.60	38.80	80.46	74.61
nl	82.72	83.83	82.51	81.92	83.98	83.11	46.86	81.63	66.80	45.39	80.70	78.51
no	88.63	89.57	88.30	88.16	90.01	89.92	51.08	87.00	72.12	50.70	86.52	82.69
orv	71.51	71.11	70.47	71.11	72.35	72.29	56.67	69.19	62.92	56.75	69.46	67.29
pl	83.28	83.82	82.53	82.44	84.08	83.92	53.48	80.78	68.14	52.34	80.39	76.26
pt	89.66	89.72	89.20	88.76	90.91	90.36	39.43	86.87	67.15	38.85	85.64	82.11
ro	81.89	82.27	81.53	81.39	83.20	82.84	48.62	80.51	66.76	48.19	79.15	76.39
ru	83.47	85.15	84.01	83.46	85.54	85.54	51.14	81.88	67.77	49.95	82.41	78.20
sk	83.73	84.22	83.90	83.24	84.68	85.02	54.35	79.49	66.42	53.15	80.25	76.97
sl	82.44	83.01	81.82	81.60	84.55	84.38	50.21	79.16	61.64	47.34	79.82	75.74
sr	86.14	86.92	84.93	84.45	87.80	87.72	49.48	83.76	63.69	50.39	83.78	79.61
sv	85.24	85.70	84.81	84.70	86.91	86.28	43.41	80.97	61.26	43.87	81.06	75.53
te	90.04	89.18	89.61	89.18	91.77	88.74	85.28	85.28	86.15	80.95	89.61	86.58
tr	80.06	83.05	80.36	79.18	83.73	83.97	54.20	79.30	62.92	53.75	78.10	74.75
ug	84.69	86.80	85.02	83.21	87.46	87.83	55.43	83.53	65.29	52.68	83.30	81.09
uk	85.22	85.61	84.46	84.14	86.70	86.62	48.78	80.84	62.84	47.49	79.61	76.04
ur	87.21	88.06	87.72	87.59	87.91	88.48	49.31	85.42	66.43	46.97	84.05	80.20
vi	81.43	84.40	81.50	81.09	84.78	84.49	46.60	79.75	58.78	46.08	80.32	75.76
wo	86.77	87.39	85.92	85.49	88.67	88.43	42.78	81.16	57.11	41.56	80.71	74.91
zh	87.19	87.62	86.76	87.23	88.10	88.07	46.10	83.42	63.46	45.22	82.53	77.05
AVG	84.21	85.03	83.86	83.63	85.63	85.56	49.49	81.28	65.65	48.44	81.33	77.53

Table A.3: Detailed EMS scores for the experiment on encoders in §4.2.

	sub					full				
	Rand	L2R	R2L	H2D	TSP	Rand	L2R	R2L	TSP	Swap
af	10.41	83.82	82.96	84.38	84.53	1.54	77.25	78.57	81.18	57.17
ar	12.50	85.71	84.98	85.01	85.75	1.13	79.33	77.05	80.85	58.84
bg	12.50	83.90	82.84	83.85	84.65	3.29	78.84	79.69	80.94	62.64
ca	11.13	86.59	86.36	86.83	87.71	1.67	79.03	80.59	81.90	61.69
cs	11.52	74.15	74.20	72.69	72.19	2.26	69.17	69.57	72.77	48.62
cu	14.05	61.61	61.36	63.33	63.53	6.50	58.29	58.16	59.48	46.31
da	10.49	78.47	77.63	78.52	78.78	1.90	74.27	73.87	76.58	56.19
de	10.41	76.08	75.68	76.22	77.12	3.08	72.09	72.03	74.47	60.14
el	12.48	81.90	81.94	82.51	83.54	3.12	73.73	74.39	77.22	55.55
en	11.97	85.60	84.75	85.48	85.58	2.87	81.44	80.66	82.42	66.52
es	11.22	84.44	84.11	84.54	85.04	1.85	74.22	79.54	80.04	56.32
et	11.43	69.36	68.67	68.71	70.41	2.85	63.39	63.88	65.30	49.47
eu	11.57	59.77	60.45	60.00	60.62	3.26	58.12	60.52	61.63	45.18
fa	11.32	83.79	83.68	84.30	84.84	1.12	78.64	80.00	81.24	64.87
fi	13.20	77.38	77.49	77.47	77.95	5.03	76.14	76.19	76.56	63.46
fr	11.38	89.50	89.27	89.60	90.20	2.50	87.28	86.45	86.50	68.78
fro	12.02	66.22	67.30	65.63	68.31	4.35	66.27	66.34	67.27	53.00
gl	11.52	81.59	81.50	82.05	82.42	2.10	73.99	75.00	76.72	54.91
got	13.15	56.76	56.77	57.07	58.99	4.75	54.64	53.34	54.29	41.97
grc	12.39	54.41	54.94	55.75	56.93	4.56	51.78	51.94	53.38	39.75
he	12.32	84.02	83.52	83.85	84.25	2.61	77.05	77.49	79.05	58.51
hi	10.52	82.97	83.53	83.06	84.13	1.69	82.03	83.64	83.87	71.67
hr	11.09	78.21	78.47	78.62	80.83	1.53	71.08	72.94	74.81	55.79
hu	10.72	62.70	62.51	63.32	64.61	1.82	52.45	53.61	57.04	33.66
hy	12.28	67.06	66.74	68.17	66.51	2.95	53.65	53.27	59.11	36.97
id	12.76	82.96	82.64	82.12	83.83	2.01	75.31	74.84	76.97	55.94
it	11.79	84.71	84.12	84.35	85.08	3.34	79.26	80.48	80.78	61.82
ja	10.50	90.64	90.68	90.43	91.08	2.03	86.71	86.94	86.59	66.99
ko	13.67	77.54	78.51	79.00	80.03	2.95	79.71	80.97	81.49	68.78
la	11.09	57.61	57.97	58.44	59.83	3.16	54.07	56.22	58.23	45.33
lt	12.46	68.00	66.39	69.18	69.59	3.01	51.01	53.20	58.05	33.56
lv	11.42	72.81	73.30	72.83	75.01	2.31	63.74	67.13	68.38	50.79
lzh	16.47	87.98	87.01	87.13	87.84	8.71	86.28	86.93	86.97	78.87
mt	11.35	79.18	79.45	80.89	80.36	1.91	69.05	68.67	71.81	47.98
nl	9.89	75.08	73.70	75.57	75.77	2.20	72.84	72.57	73.51	57.21
no	10.87	85.01	84.58	85.31	85.92	2.49	82.00	82.17	82.65	66.26
orv	13.25	57.32	56.50	57.29	59.89	6.14	56.25	55.52	55.94	41.99
pl	12.06	74.92	74.72	75.23	77.27	2.67	70.70	71.74	73.01	55.47
pt	11.15	87.80	87.97	87.78	88.95	2.13	83.05	82.90	83.82	64.54
ro	11.86	76.43	77.06	76.57	78.34	2.81	72.72	73.92	75.06	60.39
ru	12.10	77.46	78.34	78.57	79.62	2.36	72.70	74.29	76.32	55.97
sk	12.26	72.53	72.59	73.65	74.03	2.67	67.03	67.28	68.45	45.81
sl	11.33	74.63	74.66	74.72	76.33	2.31	70.15	70.78	71.62	54.16
sr	11.80	82.05	82.14	81.82	83.11	1.68	76.62	77.37	78.83	59.77
sv	10.34	79.35	79.20	79.06	80.30	2.40	71.10	70.88	73.74	53.13
te	12.90	88.51	87.56	86.71	89.08	9.94	82.82	82.39	84.74	77.10
tr	13.37	70.24	69.21	71.29	72.15	4.76	63.20	63.47	66.02	48.80
ug	13.83	77.47	77.34	78.22	78.83	3.91	70.85	70.66	72.87	57.45
uk	11.38	75.61	75.52	76.37	78.53	1.58	66.79	67.66	69.92	47.94
ur	10.31	80.15	80.39	80.59	80.62	1.19	76.84	77.58	78.33	61.85
vi	11.93	76.11	75.78	77.25	78.25	2.53	68.88	69.46	71.88	53.93
wo	11.37	83.56	83.26	84.11	83.66	1.85	72.09	71.97	74.27	54.73
zh	11.42	83.37	83.45	83.20	83.27	1.62	72.60	75.59	76.04	53.70
AVG	11.86	76.89	76.71	77.14	78.04	2.96	71.29	71.93	73.60	55.63

Table A.4: Detailed BLEU scores for the experiment on decoders in §4.3.

	sub					full				
	Rand	L2R	R2L	H2D	TSP	Rand	L2R	R2L	TSP	Swap
af	0.00	29.90	26.29	31.96	32.47	0.00	30.93	27.32	28.87	13.40
ar	0.77	33.00	28.60	30.25	33.44	0.22	27.83	24.09	28.16	14.63
bg	0.54	51.30	47.26	52.38	51.57	0.63	47.98	47.71	49.96	34.62
ca	0.06	31.66	30.72	33.18	37.10	0.00	26.10	26.16	27.74	13.81
cs	3.24	32.87	32.45	30.68	29.75	3.01	31.10	31.74	33.09	20.50
cu	5.96	39.61	38.96	40.63	40.07	6.15	42.59	42.50	40.07	37.65
da	3.01	41.31	39.72	39.89	40.96	3.19	40.43	39.72	42.20	28.01
de	0.25	36.80	36.30	35.92	39.55	0.13	36.30	33.79	38.05	26.53
el	1.49	26.55	27.05	26.80	28.29	1.74	21.59	20.84	21.59	12.90
en	9.84	66.53	64.84	65.48	65.83	9.59	63.39	63.34	63.39	53.30
es	0.36	25.94	24.37	26.72	29.81	0.30	18.08	21.95	22.01	12.09
et	1.89	32.06	31.01	31.07	32.96	1.86	28.86	30.27	30.18	21.79
eu	0.28	19.91	22.08	20.58	20.08	0.00	22.30	23.53	23.53	14.24
fa	0.67	37.90	36.89	37.90	40.23	0.50	30.88	34.22	34.72	23.87
fi	5.92	61.55	61.44	61.33	61.28	5.49	62.08	62.88	63.15	52.27
fr	0.00	51.08	50.47	51.22	53.39	0.00	47.76	46.82	46.07	28.39
fro	3.96	38.87	41.04	39.58	42.45	3.47	45.22	46.36	44.84	37.08
gl	0.12	16.74	14.88	18.49	20.00	0.12	10.35	10.70	11.51	3.84
got	5.38	31.68	30.76	31.47	33.40	3.55	34.31	34.42	33.81	29.34
grc	2.55	22.37	22.37	21.39	24.14	3.53	25.91	24.93	26.79	20.71
he	0.00	35.95	34.71	36.36	36.57	0.83	30.17	28.51	28.51	17.15
hi	0.06	39.00	39.30	38.16	41.29	0.00	39.30	41.29	42.56	27.61
hr	0.10	24.38	24.48	25.21	28.54	0.10	20.21	22.60	22.40	12.50
hu	0.00	6.35	6.12	6.35	9.98	0.00	4.76	4.08	4.08	2.72
hy	1.05	24.91	23.34	25.26	22.65	1.39	19.86	20.21	22.82	11.85
id	0.18	44.90	41.32	43.11	45.80	0.36	37.39	36.31	38.64	24.15
it	0.18	45.21	41.13	44.15	46.28	0.18	36.52	39.54	40.60	28.01
ja	0.39	57.73	58.71	57.93	57.53	0.20	53.03	50.88	51.27	30.72
ko	0.44	44.68	46.22	45.84	48.16	0.29	51.74	51.40	53.10	39.40
la	3.66	23.42	23.08	23.47	24.46	3.57	24.42	25.65	24.94	21.47
lt	1.29	22.82	22.82	22.49	26.21	1.29	19.58	17.15	21.68	11.17
lv	1.97	32.43	31.44	32.80	35.57	1.36	28.98	30.76	33.85	23.37
lzh	15.93	90.57	89.74	89.20	89.99	15.97	89.40	89.94	89.74	82.12
mt	0.92	27.02	24.71	29.56	27.25	0.46	22.17	21.02	21.25	11.09
nl	0.42	36.49	33.70	36.35	35.10	0.28	35.52	35.79	34.82	21.87
no	3.78	58.16	54.96	58.61	59.73	2.99	56.79	56.00	55.13	43.30
orv	4.27	38.12	36.39	38.01	41.52	4.48	40.60	40.77	40.17	32.18
pl	0.36	34.76	32.60	33.09	36.92	0.50	32.78	33.90	35.25	25.21
pt	0.00	40.99	39.26	40.25	45.21	0.08	36.86	35.37	36.28	21.82
ro	1.81	34.98	37.26	36.22	38.50	1.52	35.65	35.84	37.74	27.00
ru	0.77	35.34	35.02	35.04	38.35	0.96	33.96	35.83	36.59	24.00
sk	7.26	44.43	43.40	45.57	45.09	6.13	40.75	42.74	42.08	30.00
sl	0.41	26.29	25.20	24.80	30.11	0.82	25.48	25.89	24.25	16.35
sr	0.37	32.65	30.04	31.34	32.84	0.19	28.54	28.54	25.56	14.18
sv	0.60	38.29	37.70	36.51	39.09	0.79	33.13	32.74	32.94	22.42
te	4.58	87.02	84.73	83.97	86.26	3.05	81.68	80.15	83.21	77.10
tr	3.04	48.38	45.75	47.77	49.49	3.34	46.05	43.93	46.86	38.56
ug	0.89	49.00	46.89	49.33	49.44	0.22	45.11	43.22	45.89	34.67
uk	1.34	31.99	31.70	31.85	37.20	1.34	25.89	28.87	28.27	20.24
ur	0.00	27.72	28.62	28.44	27.54	0.00	26.45	26.63	27.36	17.03
vi	0.62	40.62	37.88	40.12	42.50	0.38	33.75	32.88	37.00	20.75
wo	0.22	38.75	37.86	39.64	38.98	0.22	30.96	30.96	29.40	19.60
zh	0.00	34.20	28.60	30.40	28.20	0.00	23.20	24.60	19.80	11.40
AVG	1.95	38.21	37.02	38.00	39.61	1.83	35.56	35.61	36.30	25.66

Table A.5: Detailed EMF scores for the experiment on decoders in §4.3.

	sub					full				
	Rand	L2R	R2L	H2D	TSP	Rand	L2R	R2L	TSP	Swap
af	21.18	90.44	89.66	90.78	91.13	21.23	84.17	85.44	86.52	74.41
ar	25.86	91.88	91.02	91.33	92.08	25.55	87.96	86.22	88.50	78.34
bg	21.62	88.17	85.95	88.40	88.53	22.02	83.45	83.95	84.88	75.69
ca	20.04	89.40	88.56	89.83	90.66	19.91	82.87	84.24	84.88	72.37
cs	23.87	83.33	83.17	82.53	82.32	23.48	78.75	78.90	81.21	64.72
cu	26.59	72.75	72.70	74.72	74.92	27.37	70.31	70.46	71.59	63.55
da	22.11	87.09	86.42	86.89	86.92	21.31	81.32	81.02	81.77	71.74
de	21.41	83.97	83.81	83.83	84.59	21.64	79.97	79.92	81.37	73.18
el	18.72	85.93	85.70	86.53	87.50	17.98	79.54	79.30	81.26	68.78
en	20.94	89.57	88.87	89.50	89.60	20.51	85.90	85.24	86.09	78.02
es	19.11	87.35	86.32	87.60	88.46	19.25	78.49	82.53	83.17	67.41
et	25.66	79.38	78.46	78.86	80.18	24.96	74.84	75.39	75.94	66.75
eu	29.06	77.33	77.40	77.28	78.26	28.14	72.78	74.37	74.71	66.01
fa	26.41	91.85	91.46	91.95	92.38	26.94	88.08	88.91	89.84	82.55
fi	25.96	86.88	87.15	87.05	87.42	25.85	85.71	85.75	85.55	78.70
fr	18.01	91.38	91.05	91.63	91.93	17.90	88.73	88.31	88.27	74.89
fro	23.44	77.59	78.57	77.34	79.95	22.66	76.65	76.84	77.10	68.45
gl	17.99	85.24	84.52	85.23	86.01	17.90	78.53	79.65	80.38	67.12
got	25.95	70.81	71.13	71.03	72.85	25.48	69.11	68.35	68.99	61.78
grc	26.27	69.72	69.66	70.83	72.32	26.48	66.87	66.96	67.77	60.58
he	24.11	90.60	89.36	90.49	90.47	24.30	85.26	85.57	86.40	75.66
hi	22.18	90.42	90.30	90.10	91.04	22.58	87.96	88.93	89.09	82.41
hr	24.02	85.06	84.84	85.63	87.10	24.83	80.17	80.86	81.91	71.69
hu	26.57	79.70	79.05	80.20	81.18	25.29	72.30	72.23	73.73	60.73
hy	27.24	78.59	78.62	79.75	78.27	25.45	70.14	69.17	72.49	60.45
id	23.73	89.08	88.21	88.29	89.36	23.48	82.14	81.41	82.56	70.89
it	18.19	87.95	86.89	87.38	88.33	17.72	83.20	84.12	84.53	70.30
ja	19.40	92.86	92.93	92.76	93.25	20.12	90.32	89.66	89.51	79.73
ko	33.30	91.05	91.44	91.80	92.38	33.20	90.66	90.96	91.25	85.27
la	23.75	70.05	70.32	70.91	71.66	23.77	65.51	66.78	68.42	60.65
lt	28.33	79.86	78.60	80.70	80.90	27.54	67.95	69.82	72.14	60.94
lv	23.88	80.32	79.83	80.38	82.02	23.43	73.19	74.87	75.98	67.47
lzh	30.07	95.41	94.86	94.70	95.16	30.48	94.77	95.14	94.89	91.74
mt	21.68	85.67	84.82	86.12	85.90	22.63	78.05	78.47	80.46	68.98
nl	18.94	83.78	82.28	83.88	83.98	18.68	80.68	80.08	80.70	70.14
no	21.55	89.43	88.43	89.82	90.01	20.97	86.33	86.11	86.52	75.36
orv	25.77	70.05	68.62	70.34	72.35	25.99	69.14	68.70	69.46	60.15
pl	25.06	82.94	81.64	82.47	84.08	24.44	79.16	79.87	80.39	68.98
pt	17.70	89.95	89.34	89.82	90.91	18.04	85.15	85.13	85.64	74.14
ro	21.95	81.59	81.87	81.84	83.20	22.07	77.32	78.49	79.15	70.45
ru	23.82	84.48	83.89	84.56	85.54	24.17	79.78	80.22	82.41	69.98
sk	25.45	83.65	83.26	84.55	84.68	25.45	78.69	79.70	80.25	65.70
sl	23.05	82.97	82.60	83.57	84.55	22.48	78.22	79.22	79.82	70.14
sr	23.66	87.41	86.51	87.31	87.80	23.18	82.72	83.26	83.78	73.55
sv	21.20	86.85	86.28	86.68	86.91	21.66	79.71	79.83	81.06	68.71
te	20.35	91.77	90.48	89.61	91.77	22.94	87.45	86.58	89.61	85.71
tr	27.25	82.60	81.45	82.56	83.73	26.10	78.01	77.35	78.10	72.04
ug	28.94	86.90	85.85	86.99	87.46	28.96	83.77	82.48	83.30	76.19
uk	23.84	84.32	83.81	85.24	86.70	24.29	77.48	78.17	79.61	67.66
ur	22.03	87.48	87.48	87.91	87.91	21.72	84.59	84.31	84.05	74.24
vi	25.71	83.88	83.29	84.20	84.78	24.31	79.00	79.86	80.32	73.56
wo	21.14	88.35	87.87	88.88	88.67	21.94	79.64	79.80	80.71	72.07
zh	26.42	88.83	87.95	88.01	88.10	25.27	81.08	82.22	82.53	70.32
AVG	23.59	84.79	84.24	84.92	85.63	23.47	80.07	80.40	81.33	71.34

Table A.6: Detailed EMS scores for the experiment on decoders in §4.3.

	d100	d200	d400	d800	d1600	d3200	d6400
af	77.72	80.19	82.48	84.26	84.83	85.02	83.86
ar	74.47	78.50	80.58	81.38	83.75	85.17	85.85
bg	59.56	67.67	72.28	74.05	77.37	79.09	83.67
ca	76.81	78.61	80.42	82.79	84.24	85.63	86.36
cs	57.46	60.85	62.23	64.51	67.90	69.94	71.05
cu	45.81	50.01	53.43	56.74	59.25	63.20	64.12
da	65.58	68.99	71.76	74.32	76.60	78.12	78.47
de	62.30	63.98	65.42	68.22	72.23	75.69	76.54
el	75.95	78.22	79.71	81.31	83.72	83.66	83.48
en	73.83	76.56	78.84	80.72	82.65	83.87	84.89
es	74.26	76.79	77.96	80.44	80.83	82.76	84.08
et	50.63	54.04	57.35	60.75	62.98	65.20	66.17
eu	46.17	49.93	52.39	53.87	55.96	58.33	60.01
fa	73.41	74.19	75.77	79.18	82.96	84.70	85.16
fi	54.65	57.26	62.93	67.44	71.87	74.16	76.42
fr	81.69	83.80	85.33	87.23	88.40	88.45	89.78
fro	54.62	54.37	56.52	59.84	59.54	63.63	67.15
gl	74.06	77.02	79.53	80.09	81.86	82.77	82.62
got	45.29	48.25	50.88	53.38	57.42	59.74	59.63
grc	38.87	38.96	39.46	41.50	42.80	46.94	53.97
he	69.99	76.11	78.19	80.72	82.45	83.38	84.06
hi	69.16	73.98	76.57	78.35	80.62	82.13	83.15
hr	64.96	68.42	71.57	74.81	76.60	78.25	80.08
hu	52.97	56.58	58.40	63.50	63.63	63.59	63.81
hy	57.43	60.05	63.49	67.55	66.96	67.53	67.39
id	72.27	75.97	78.19	80.10	81.96	83.58	84.35
it	70.50	74.16	76.83	79.21	81.12	81.54	84.22
ja	86.06	87.36	88.47	88.74	89.94	90.63	91.02
ko	68.40	70.20	70.58	73.52	74.02	75.42	77.74
la	41.49	43.50	47.67	51.35	53.75	56.72	58.59
lt	58.93	63.21	65.67	68.18	69.60	71.15	69.51
lv	57.46	61.00	62.23	64.20	68.82	71.75	73.28
lzh	73.74	77.34	79.04	82.94	84.11	86.92	87.25
mt	72.29	73.36	75.88	78.90	79.92	80.36	81.04
nl	63.36	66.34	68.91	70.35	72.29	73.53	74.07
no	69.16	73.09	76.07	78.27	80.25	83.12	84.70
orv	35.42	35.76	37.68	42.72	49.53	54.55	57.95
pl	53.20	54.30	54.11	56.09	57.39	72.01	74.41
pt	76.69	80.81	82.39	85.24	86.47	87.25	87.89
ro	62.73	67.31	70.28	72.37	75.57	76.53	77.81
ru	61.03	61.07	65.38	67.27	72.29	74.15	76.52
sk	60.03	64.29	66.08	70.56	71.76	73.91	74.45
sl	60.66	64.60	67.57	70.43	72.70	74.38	76.24
sr	69.20	74.30	76.56	79.85	81.32	82.79	83.08
sv	65.09	67.58	70.16	72.00	75.23	78.83	80.09
te	81.01	81.02	83.92	84.33	87.19	85.77	84.48
tr	56.76	62.72	64.27	66.66	70.61	71.70	72.57
ug	69.87	72.22	75.47	77.61	79.18	78.73	78.82
uk	64.50	66.35	69.79	71.55	75.22	76.29	78.38
ur	72.40	74.97	76.48	78.01	79.62	80.77	81.12
vi	63.36	66.57	70.79	75.19	78.18	77.27	78.63
wo	69.71	73.68	80.38	82.77	83.91	83.66	83.22
zh	71.48	74.40	76.45	77.89	80.84	82.81	83.23
AVG	64.23	67.19	69.64	72.14	74.34	76.17	77.29

Table A.7: Detailed BLEU scores for the experiment on training data sizes in §4.6.

	100	200	400	800	1600	3200	6400
af	20.10	26.29	32.47	31.44	32.47	32.47	28.87
ar	15.51	19.80	21.45	23.21	28.71	32.67	35.53
bg	18.65	25.38	31.57	33.63	41.17	42.42	49.87
ca	17.50	18.61	22.06	26.57	28.85	32.07	32.24
cs	15.59	17.38	16.84	20.53	24.54	26.80	27.95
cu	24.42	26.56	29.36	32.90	36.35	41.57	41.10
da	21.99	27.13	31.21	33.33	36.88	40.96	41.31
de	20.53	21.90	22.78	27.91	31.91	37.17	38.42
el	19.60	21.84	22.83	25.06	29.28	29.78	27.30
en	46.20	49.05	52.65	54.55	58.04	61.89	63.54
es	14.63	16.93	16.75	21.10	21.77	24.49	27.27
et	14.21	15.55	18.24	22.05	23.42	25.57	26.53
eu	9.84	12.68	13.85	15.29	17.35	19.80	20.30
fa	18.53	20.03	20.87	24.54	32.55	35.89	40.23
fi	33.60	35.47	43.15	47.68	51.89	55.95	59.89
fr	33.27	37.80	41.33	45.46	47.36	48.58	52.10
fro	26.06	25.84	30.13	31.32	33.33	38.27	42.35
gl	8.49	10.23	13.95	15.00	17.21	19.07	20.12
got	19.49	22.23	25.18	26.80	31.98	34.01	32.99
grc	8.15	7.26	8.73	9.91	9.52	11.78	19.82
he	14.26	22.52	25.83	27.07	31.20	33.88	34.71
hi	16.76	23.03	26.76	28.93	33.39	36.11	39.60
hr	11.15	13.85	16.88	20.94	22.81	24.58	27.40
hu	3.85	5.67	4.08	7.71	7.26	6.35	7.71
hy	16.03	17.42	20.21	24.74	23.00	25.61	23.87
id	25.22	29.87	34.88	35.60	40.07	44.54	47.41
it	16.49	20.92	25.00	30.14	34.57	34.75	40.25
ja	44.81	47.75	50.68	52.64	55.77	56.36	56.75
ko	31.99	33.40	34.22	38.53	39.55	42.26	44.29
la	12.04	13.37	15.99	18.94	19.94	22.23	23.56
lt	15.53	18.93	20.23	24.27	24.11	27.02	25.40
lv	18.43	22.75	19.85	22.75	28.48	32.43	33.72
lzh	73.23	76.99	78.90	83.63	85.64	89.11	89.20
mt	17.78	19.86	20.79	25.64	29.10	29.10	30.48
nl	19.92	23.40	27.44	29.11	30.64	32.17	33.98
no	30.47	37.90	41.47	45.37	48.03	54.30	56.50
orv	14.63	14.58	17.76	22.35	30.29	35.85	38.98
pl	13.69	14.54	14.41	16.52	16.75	30.80	33.81
pt	19.50	26.12	29.42	35.54	38.84	42.40	42.31
ro	20.44	24.43	28.80	31.56	35.46	36.98	38.69
ru	15.84	17.30	20.15	22.05	27.75	29.88	33.60
sk	30.57	33.58	35.85	40.38	41.60	46.32	46.60
sl	9.95	12.81	16.62	20.84	23.30	25.75	28.47
sr	13.06	20.15	23.69	30.22	30.60	32.46	34.14
sv	18.06	20.44	20.63	24.80	29.17	34.92	35.91
te	79.39	80.92	80.92	80.92	84.73	82.44	81.68
tr	34.82	39.78	38.46	42.11	47.37	48.68	49.49
ug	37.22	40.22	46.00	49.67	50.44	50.56	49.78
uk	15.77	20.39	22.17	25.30	28.42	32.89	36.90
ur	15.76	17.93	20.29	22.28	26.09	27.36	29.17
vi	20.00	24.38	28.88	35.12	41.75	40.12	42.50
wo	16.93	21.16	33.85	37.86	40.09	38.31	38.53
zh	14.60	17.40	19.60	20.20	23.20	28.20	28.00
AVG	21.97	25.13	27.85	30.98	34.04	36.72	38.32

Table A.8: Detailed EMF scores for the experiment on training data sizes in §4.6.

	100	200	400	800	1600	3200	6400
af	85.49	87.16	89.26	90.88	91.47	91.18	90.49
ar	85.32	88.02	89.07	89.57	91.00	91.74	92.25
bg	70.22	76.32	79.03	80.33	82.51	83.97	87.84
ca	82.89	84.32	85.37	87.25	88.18	89.12	89.56
cs	71.97	74.28	75.03	76.57	79.29	80.66	81.58
cu	60.29	64.48	66.62	69.02	71.52	74.75	74.85
da	77.03	79.86	82.24	84.34	85.70	86.89	86.76
de	72.47	73.74	74.89	76.66	80.93	83.97	84.22
el	81.81	83.56	84.68	85.57	87.71	87.86	87.68
en	80.16	81.59	84.08	84.88	86.88	88.02	88.73
es	80.19	82.44	82.72	84.94	85.38	86.79	87.63
et	66.90	69.25	71.45	73.69	75.32	76.41	77.30
eu	66.71	68.80	72.14	73.35	74.34	76.59	77.89
fa	85.25	86.25	87.32	89.29	91.05	92.19	92.51
fi	69.94	71.40	76.71	79.60	82.97	84.77	86.45
fr	85.17	86.93	88.09	89.54	90.43	90.47	91.64
fro	68.39	69.20	70.85	73.24	73.14	76.47	78.93
gl	79.41	81.65	83.82	83.98	85.52	86.36	86.20
got	62.44	64.66	66.45	68.22	71.94	73.46	72.85
grc	56.82	58.14	57.71	60.11	61.12	64.61	70.47
he	81.49	85.30	86.60	88.06	89.36	89.57	90.47
hi	80.38	83.37	85.38	86.89	88.68	89.55	90.32
hr	76.57	78.69	80.70	82.81	83.92	85.28	86.51
hu	74.64	76.51	77.99	80.83	80.63	80.50	80.53
hy	71.70	73.81	76.85	79.07	78.75	79.17	79.28
id	81.25	83.78	85.19	86.73	88.07	89.22	89.87
it	76.59	79.33	81.90	83.46	85.21	85.54	87.48
ja	89.46	90.75	91.09	91.42	92.33	92.78	93.13
ko	86.01	86.84	87.03	88.72	89.05	89.75	91.04
la	57.25	59.03	62.23	65.45	66.92	69.90	71.36
lt	73.85	76.65	78.30	79.94	80.55	81.79	80.82
lv	69.22	72.72	72.91	74.31	77.59	79.85	80.87
lzh	85.55	87.69	89.22	91.92	92.83	94.75	94.82
mt	80.26	80.79	82.76	85.20	86.02	86.15	86.42
nl	74.92	77.06	78.97	79.98	81.04	81.97	82.82
no	76.43	79.84	82.44	84.05	85.38	87.79	88.91
orv	51.79	52.38	55.12	59.27	64.98	68.47	71.00
pl	65.73	67.00	66.53	67.76	69.07	80.21	81.96
pt	80.97	84.28	85.49	87.93	89.02	89.81	89.84
ro	71.40	74.50	76.84	78.62	81.06	81.80	82.55
ru	71.96	72.12	75.29	76.96	80.45	81.81	83.37
sk	74.62	77.98	79.06	82.36	82.67	84.39	84.98
sl	73.32	75.90	78.07	80.35	81.91	82.87	84.49
sr	78.81	81.94	83.69	85.86	86.44	87.59	87.85
sv	75.82	77.88	79.66	80.69	83.47	85.90	86.76
te	85.71	86.58	87.88	87.88	90.48	89.61	88.74
tr	73.83	77.63	78.64	80.36	82.58	83.55	83.78
ug	82.52	83.86	85.83	86.88	87.78	87.57	87.51
uk	76.31	77.27	79.85	81.18	84.59	85.33	86.68
ur	81.09	83.01	84.46	85.76	87.01	87.97	88.23
vi	73.99	76.51	79.57	82.65	85.01	84.33	84.81
wo	77.45	80.87	86.34	87.92	88.83	88.86	88.38
zh	81.65	83.21	84.15	85.39	86.51	87.93	88.10
AVG	75.50	77.68	79.50	81.28	82.92	84.30	85.09

Table A.9: Detailed EMS scores for the experiment on training data sizes in §4.6.

Bibliography

Aharoni, R. and Goldberg, Y. (2017). Morphological Inflection Generation with Hard Monotonic Attention. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2004–2015. Cited on page 95.

Applegate, D., Bixby, R., Chvatal, V., and Cook, W. (2006). Concorde TSP Solver. Cited on page 41.

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*. Cited on page 21.

Banarescu, L., Bonial, C., Cai, S., Georgescu, M., Griffitt, K., Hermjakob, U., Knight, K., Koehn, P., Palmer, M., and Schneider, N. (2013). Abstract Meaning Representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186, Sofia, Bulgaria. Association for Computational Linguistics. Cited on page 108.

Bangalore, S. and Rambow, O. (2000). Exploiting a Probabilistic Hierarchical Model for Generation. In *Proceedings of the 18th conference on Computational linguistics-Volume 1*, pages 42–48. Association for Computational Linguistics. Cited on page 15.

Behaghel, O. (1932). *Deutsche Syntax: eine geschichtliche Darstellung. Wortstellung, Periodenbau*. Winter. Cited on page 25.

Belz, A., White, M., Espinosa, D., Kow, E., Hogan, D., and Stent, A. (2011). The First Surface Realisation Shared Task: Overview and Evaluation Results. In *Proceedings of the 13th European Workshop on Natural Language Generation*, pages 217–226, Nancy, France. Association for Computational Linguistics. Cited on pages 27 and 89.

Bengio, Y., Ducharme, R., Vincent, P., and Janvin, C. (2003). A neural probabilistic language model. *The journal of machine learning research*, 3:1137–1155. Cited on page 19.

Björkelund, A., Falenska, A., Yu, X., and Kuhn, J. (2017). IMS at the CoNLL 2017 UD shared task: CRFs and perceptrons meet neural networks. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 40–51, Vancouver, Canada. Association for Computational Linguistics.

Björkelund, A. and Kuhn, J. (2014). Learning Structured Perceptrons for Coreference Resolution with Latent Antecedents and Non-Local Features. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 47–57. Cited on pages 37 and 38.

Blohm, M., Jagfeld, G., Sood, E., Yu, X., and Vu, N. T. (2018). Comparing attention-based convolutional and recurrent neural networks: Success and limitations in machine reading comprehension. In *Proceedings of the 22nd Conference on Computational Natural Language Learning*, pages 108–118, Brussels, Belgium. Association for Computational Linguistics.

Bohnet, B., Björkelund, A., Kuhn, J., Seeker, W., and Zarriess, S. (2012). Generating Non-Projective Word Order in Statistical Linearization. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 928–939, Jeju Island, Korea. Association for Computational Linguistics. Cited on pages 1, 17, and 46.

Bohnet, B., Wanner, L., Mille, S., and Burga, A. (2010). Broad Coverage Multilingual Deep Sentence Generation with a Stochastic Multi-Level Realizer. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 98–106. Association for Computational Linguistics. Cited on pages 16, 17, 34, and 35.

Brew, C. (1992). Letting the cat out of the bag: Generation for shake-and-bake MT. In *COLING 1992 Volume 2: The 15th International Conference on Computational Linguistics*. Cited on page 15.

Brown, P. F., Cocke, J., Della Pietra, S. A., Della Pietra, V. J., Jelinek, F., Lafferty, J. D., Mercer, R. L., and Roossin, P. S. (1990). A statistical approach to machine translation. *Computational Linguistics*, 16(2):79–85. Cited on page 15.

Carroll, J., Copestake, A., Flickinger, D., and Poznanski, V. (1999). An Efficient Chart Generator for (Semi-) Lexicalist Grammars. In *Proceedings of the 7th European workshop on natural language generation (EWNLG'99)*, pages 86–95. Cited on page 15.

Chen, D. and Manning, C. (2014). A Fast and Accurate Dependency Parser Using Neural Networks. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 740–750. Cited on page 19.

Chen, X. and Gerdes, K. (2018). How do universal dependencies distinguish language groups? In Jiang, J. and Liu, H., editors, *Quantitative Analysis of Dependency Structures*, pages 277–293, Berlin/Boston. De Gruyter Mouton. Cited on page 25.

Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics. Cited on page 21.

Chomsky, N. (1957). *Syntactic Structures*. Mouton and Co., The Hague. Cited on page 9.

Chu, Y. and Liu, T. (1965). On the shortest aborescence of a directed graph. *Science Sinica*, 14:1396–1400. Cited on page 13.

Collins, M. (2002). Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002)*, pages 1–8. Association for Computational Linguistics. Cited on page 18.

Collins, M. and Roark, B. (2004). Incremental Parsing with the Perceptron Algorithm. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 111. Association for Computational Linguistics. Cited on page 38.

Coughlin, D. (2003). Correlating automated and human assessments of machine translation quality. In *Proceedings of MT summit IX*, pages 63–70. Citeseer. Cited on page 53.

de Marneffe, M.-C., Dozat, T., Silveira, N., Haverinen, K., Ginter, F., Nivre, J., and Manning, C. D. (2014). Universal Stanford dependencies: A cross-linguistic typology. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 4585–4592, Reykjavik, Iceland. European Language Resources Association (ELRA). Cited on page 63.

Dönicke, T., Yu, X., and Kuhn, J. (2020a). Identifying and handling cross-treebank inconsistencies in UD: A pilot study. In *Proceedings of the Fourth Workshop on Universal Dependencies (UDW 2020)*, pages 67–75, Barcelona, Spain (Online). Association for Computational Linguistics.

Dönicke, T., Yu, X., and Kuhn, J. (2020b). Real-valued logics for typological universals: Framework and application. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 3990–4003, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Dozat, T. and Manning, C. D. (2016). Deep Biaffine Attention for Neural Dependency Parsing. *ArXiv*, abs/1611.01734. Cited on pages 40 and 41.

Dušek, O., Novikova, J., and Rieser, V. (2018). Findings of the E2E NLG Challenge. In *Proc. of the 11th International Conference on Natural Language Generation*, pages 322–328, Tilburg, The Netherlands. Association for Computational Linguistics. Cited on page 108.

Dyer, C., Ballesteros, M., Ling, W., Matthews, A., and Smith, N. A. (2015). Transition-based dependency parsing with stack long short-term memory. 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 334–343, Beijing, China. Association for Computational Linguistics. Cited on page 44.

Edmonds, J. (1967). Optimum branchings. *Journal of Research of the National Bureau of Standards*, 71(B):233–240. Cited on page 13.

Eisner, J. M. (1996). Three New Probabilistic Models for Dependency Parsing: An Exploration. In *COLING 1996 Volume 1: The 16th International Conference on Computational Linguistics*. Cited on page 13.

Elder, H., Burke, R., O'Connor, A., and Foster, J. (2020). Shape of synth to come: Why we should use synthetic data for English surface realization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7465–7471, Online. Association for Computational Linguistics. Cited on pages 30 and 97.

Elder, H. and Hokamp, C. (2018). Generating High-Quality Surface Realizations Using Data Augmentation and Factored Sequence Models. In *Proceedings of the First Workshop on Multilingual Surface Realisation*, pages 49–53, Melbourne, Australia. Association for Computational Linguistics. Cited on page 30.

Elhadad, M. and Robin, J. (1992). Controlling Content Realization with Functional Unification Grammars. In *International Workshop on Natural Language Generation*, pages 89–104. Springer. Cited on page 15.

Elman, J. L. (1990). Finding structure in time. *Cognitive science*, 14(2):179–211. Cited on page 20.

Ferrer-i-Cancho, R. (2006). Why do syntactic links not cross? *EPL (Europhysics Letters)*, 76(6):1228. Cited on pages 25 and 88.

Ferrer-i-Cancho, R. (2015). The placement of the head that minimizes online memory: a complex systems approach. *Language Dynamics and Change*, 5(1):114–137. Cited on page 80.

Ferrer-i-Cancho, R. and Gómez-Rodríguez, C. (2016). Crossings as a side effect of dependency lengths. *Complexity*, 21(S2):320–328. Cited on pages 25, 76, and 85.

Filippova, K. and Strube, M. (2009). Tree Linearization in English: Improving Language Model Based Approaches. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, pages 225–228. Association for Computational Linguistics. Cited on page 15.

Futrell, R., Mahowald, K., and Gibson, E. (2015a). Large-scale evidence of dependency length minimization in 37 languages. *Proceedings of the National Academy of Sciences*, 112(33):10336–10341. Cited on pages x, 2, 25, 76, 77, 85, and 88.

Futrell, R., Mahowald, K., and Gibson, E. (2015b). Quantifying Word Order Freedom in Dependency Corpora. In *Proceedings of the third international conference on dependency linguistics (Depling 2015)*, pages 91–100. Cited on page 37.

Gardent, C., Shimorina, A., Narayan, S., and Perez-Beltrachini, L. (2017). The WebNLG challenge: Generating text from RDF data. In *Proceedings of the 10th International Conference on Natural Language Generation*, pages 124–133, Santiago de Compostela, Spain. Association for Computational Linguistics. Cited on page 108.

Germann, U., Jahr, M., Knight, K., Marcu, D., and Yamada, K. (2001). Fast Decoding and Optimal Decoding for Machine Translation. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, pages 228–235, Toulouse, France. Association for Computational Linguistics. Cited on page 16.

Gibson, E. (1998). Linguistic Complexity: Locality of Syntactic Dependencies. *Cognition*, 68(1):1–76. Cited on pages 37 and 75.

Gibson, E. (2000). The dependency locality theory: A distance-based theory of linguistic complexity. *Image, language, brain*, 2000:95–126. Cited on page 75.

Gildea, D. and Temperley, D. (2007). Optimizing grammars for minimum dependency length. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 184–191. Cited on pages 76 and 77.

Gildea, D. and Temperley, D. (2010). Do grammars minimize dependency length? *Cognitive Science*, 34(2):286–310. Cited on page 25.

Gómez-Rodríguez, C. and Ferrer-i-Cancho, R. (2017). Scarcity of crossing dependencies: A direct outcome of a specific constraint? *Physical Review E*, 96(6):062304. Cited on pages 25 and 85.

Graves, A. and Schmidhuber, J. (2005). Framewise Phoneme Classification with Bidirectional LSTM and other Neural Network Architectures. *Neural networks*, 18(5-6):602–610. Cited on pages 21 and 32.

Graves, A., Wayne, G., and Danihelka, I. (2014). Neural turing machines. *arXiv preprint arXiv:1410.5401*. Cited on page 21.

Greenberg, J. H. (1963). Some universals of grammar with particular reference to the order of meaningful elements. *Universals of language*, 2:73–113. Cited on page 2.

Gulordava, K. (2018). *Word Order Variation and Dependency Length Minimisation: A Cross-Linguistic Computational Approach*. PhD thesis, University of Geneva. Cited on pages 25, 37, and 79.

Gulordava, K. and Merlo, P. (2015). Diachronic trends in word order freedom and dependency length in dependency-annotated corpora of latin and ancient greek. In *Proceedings of the third international conference on dependency linguistics (Depling 2015)*, pages 121–130. Cited on pages 25 and 66.

Gulordava, K., Merlo, P., and Crabbé, B. (2015). Dependency length minimisation effects in short spans: a large-scale analysis of adjective placement in complex noun phrases. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, volume 2, pages 477–482. Cited on page 25.

Haug, D. T. T. and Jøhndal, M. L. (2008). Creating a parallel treebank of the old indo-european bibletranslations. Cited on page 64.

Hawkins, J. A. (1992). Syntactic weight versus information structure in word order variation. In *Informationsstruktur und grammatik*, pages 196–219. Springer. Cited on page 12.

He, W., Wang, H., Guo, Y., and Liu, T. (2009). Dependency Based Chinese Sentence Realization. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, pages 809–816. Association for Computational Linguistics. Cited on page 37.

Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Comput.*, 9(8):1735–1780. Cited on page 20.

Horvat, M. and Byrne, W. (2014). A Graph-Based Approach to String Regeneration. In *Proceedings of the Student Research Workshop at the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 85–95, Gothenburg, Sweden. Association for Computational Linguistics. Cited on page 16.

Hu, D. (2019). An introductory survey on attention mechanisms in nlp problems. In *Proceedings of SAI Intelligent Systems Conference*, pages 432–448. Springer. Cited on page 22.

Huang, L., Fayong, S., and Guo, Y. (2012). Structured perceptron with inexact search. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 142–151, Montréal, Canada. Association for Computational Linguistics. Cited on page 38.

Hudson, R. A. (1984). *Word Grammar*. Basil Blackwell, Oxford, UK. Cited on page 9.

Jacob, D. (2003). Adjective position, specificity, and information structure in spanish. *Arbeitspapier Nr*, 119:71. Cited on page 66.

Jiang, J. and Liu, H. (2015). The effects of sentence length on dependency distance, dependency direction and the implications-based on a parallel English-Chinese dependency treebank. *Language Sciences*, 50:93–104. Cited on page 25.

Kann, K. and Schütze, H. (2016). MED: The LMU system for the SIGMORPHON 2016 shared task on morphological reinflection. In *Proceedings of the 14th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 62–70. Cited on page 95.

Kiperwasser, E. and Goldberg, Y. (2016). Simple and Accurate Dependency Parsing Using Bidirectional LSTM Feature Representations. *Transactions of the Association for Computational Linguistics*, 4:313–327. Cited on page 42.

Kipf, T. N. and Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*. Cited on page 24.

Knight, K. (1999). Decoding Complexity in Word-Replacement Translation Models. *Computational Linguistics*, 25(4):607–615. Cited on pages 16 and 39.

Konstas, I., Iyer, S., Yatskar, M., Choi, Y., and Zettlemoyer, L. (2017). Neural AMR: Sequence-to-sequence models for parsing and generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 146–157, Vancouver, Canada. Association for Computational Linguistics. Cited on page 30.

Kübler, S., McDonald, R., and Nivre, J. (2009). Dependency parsing. *Synthesis lectures on human language technologies*, 1(1):1–127. Cited on page 13.

Kürschner, W. (2017). *Grammatisches Kompendium: systematisches Verzeichnis grammatischer Grundbegriffe*. UTB. Cited on page 70.

Lambrecht, K. (1996). *Information structure and sentence form: Topic, focus, and the mental representations of discourse referents*, volume 71. Cambridge university press. Cited on page 12.

Langkilde, I. and Knight, K. (1998). Generation that Exploits Corpus-based Statistical Knowledge. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics-Volume 1*, pages 704–710. Association for Computational Linguistics. Cited on page 15.

Lavoie, B. and Rainbow, O. (1997). A Fast and Portable Realizer for Text Generation Systems. In *Fifth Conference on Applied Natural Language Processing*. Cited on page 15.

Liu, H. (2008). Dependency distance as a metric of language comprehension difficulty. *Journal of Cognitive Science*, 9(2):159–191. Cited on pages 2 and 25.

Liu, H. (2010). Dependency Direction as a Means of Word-Order Typology: A Method Based on Dependency Treebanks. *Lingua*, 120(6):1567–1578. Cited on page 37.

Liu, Y., Zhang, Y., Che, W., and Qin, B. (2015). Transition-Based Syntactic Linearization. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 113–122, Denver, Colorado. Association for Computational Linguistics. Cited on page 17.

Luong, M.-T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*. Cited on page 21.

McDonald, R., Pereira, F., Ribarov, K., and Hajič, J. (2005). Non-Projective Dependency Parsing using Spanning Tree Algorithms. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 523–530, Vancouver, British Columbia, Canada. Association for Computational Linguistics. Cited on pages 13 and 42.

Mel'cuk, I. A. et al. (1988). *Dependency syntax: theory and practice*. SUNY press. Cited on pages 9 and 108.

Meyers, A., Reeves, R., Macleod, C., Szekely, R., Zielinska, V., Young, B., and Grishman, R. (2004). The NomBank project: An interim report. In *Proceedings of the Workshop Frontiers in Corpus Annotation at HLT-NAACL 2004*, pages 24–31, Boston, Massachusetts, USA. Association for Computational Linguistics. Cited on pages 27 and 91.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *arXiv preprint arXiv:1310.4546*. Cited on page 19.

Mille, S., Belz, A., Bohnet, B., Castro Ferreira, T., Graham, Y., and Wanner, L. (2020). The third multilingual surface realisation shared task (SR'20): Overview and evaluation results. In *Proceedings of the Third Workshop on Multilingual Surface Realisation*, pages 1–20, Barcelona, Spain (Online). Association for Computational Linguistics. Cited on pages 27, 89, and 103.

Mille, S., Belz, A., Bohnet, B., Graham, Y., Pitler, E., and Wanner, L. (2018). The First Multilingual Surface Realisation Shared Task (SR'18): Overview and Evaluation Results. In *Proceedings of the First Workshop on Multilingual Surface Realisation*, pages 1–12, Melbourne, Australia. Association for Computational Linguistics. Cited on pages 27 and 89.

Mille, S., Belz, A., Bohnet, B., Graham, Y., and Wanner, L. (2019). The Second Multilingual Surface Realisation Shared Task (SR'19): Overview and Evaluation Results. In

Proceedings of the 2nd Workshop on Multilingual Surface Realisation (MSR 2019), pages 1–17, Hong Kong, China. Association for Computational Linguistics. Cited on pages 27, 43, 89, and 100.

Miwa, M. and Bansal, M. (2016). End-to-End Relation Extraction using LSTMs on Sequences and Tree Structures. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1105–1116. Cited on pages 23 and 31.

Neeleman, A. and Van de Koot, J. (2016). Word order and information structure. Oxford University Press. Cited on page 12.

Nivre, J. (2008). Algorithms for Deterministic Incremental Dependency Parsing. *Computational Linguistics*, 34(4):513–553. Cited on page 14.

Nivre, J. (2009). Non-Projective Dependency Parsing in Expected Linear Time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 351–359, Suntec, Singapore. Association for Computational Linguistics. Cited on pages 15, 44, and 46.

Nivre, J., Abrams, M., Agić, Ž., Ahrenberg, L., Aleksandravičiūtė, G., Antonsen, L., Aplonova, K., Aranzabe, M. J., Arutie, G., Asahara, M., Ateyah, L., Attia, M., Atutxa, A., Augustinus, L., Badmaeva, E., Ballesteros, M., Banerjee, E., Bank, S., Barbu Mititelu, V., Basmov, V., Bauer, J., Bellato, S., Bengoetxea, K., Berzak, Y., Bhat, I. A., Bhat, R. A., Biagetti, E., Bick, E., Bielinskienė, A., Blokland, R., Bobicev, V., Boizou, L., Borges Völker, E., Börstell, C., Bosco, C., Bouma, G., Bowman, S., Boyd, A., Brokaitė, K., Burchardt, A., Candito, M., Caron, B., Caron, G., Cebiroğlu Eryiğit, G., Cecchini, F. M., Celano, G. G. A., Čéplö, S., Cetin, S., Chalub, F., Choi, J., Cho, Y., Chun, J., Cinková, S., Collomb, A., Çöltekin, Ç., Connor, M., Courtin, M., Davidson, E., de Marneffe, M.-C., de Paiva, V., Diaz de Ilarraza, A., Dickerson, C., Dione, B., Dirix, P., Dobrovoljc, K., Dozat, T., Droganova, K., Dwivedi, P., Eckhoff, H., Eli, M., Elkahky, A., Ephrem, B., Erjavec, T., Etienne, A., Farkas, R., Fernandez Alcalde, H., Foster, J., Freitas, C., Fujita, K., Gajdošová, K., Galbraith, D., Garcia, M., Gärdenfors, M., Garza, S., Gerdes, K., Ginter, F., Goenaga, I., Gojenola, K., Gökırmak, M., Goldberg, Y., Gómez Guinovart, X., González Saavedra, B., Grioni, M., Grūzītis, N., Guillaume, B., Guillot-Barbance, C., Habash, N., Hajič, J., Hajič jr., J., Hà Mỹ, L., Han, N.-R., Harris, K., Haug, D., Heinecke, J., Hennig, F., Hladká, B., Hlaváčová, J., Hociung, F., Hohle, P., Hwang, J., Ikeda, T., Ion, R., Irimia, E., Ishola, Ọ., Jelínek, T., Johannsen, A., Jørgensen, F., Kaşıkara, H., Kaasen, A., Kahane, S., Kanayama, H., Kanerva, J., Katz, B., Kayadelen, T., Kenney, J., Kettnerová, V., Kirchner, J., Köhn, A., Kopacewicz, K., Kotsyba, N., Kovalevskaitė, J., Krek, S., Kwak, S., Laippala, V., Lambertino, L., Lam, L., Lando, T., Larasati, S. D., Lavrentiev, A., Lee, J., Lê H`ông, P., Lenci, A., Lertpradit, S., Leung, H., Li, C. Y., Li, J., Li, K., Lim, K., Li, Y., Ljubešić, N., Loginova, O., Lyashevskaya, O., Lynn, T., Macketanz, V., Makazhanov, A., Mandl, M., Manning, C., Manurung, R., Mărănduc, C., Mareček, D., Marheinecke, K., Martínez Alonso, H., Martins, A., Mašek, J., Matsumoto, Y., McDonald, R., McGuinness, S., Mendonça, G., Miekka, N., Misirpashayeva, M., Missilä, A., Mititelu, C., Miyao, Y., Montemagni, S., More, A., Moreno Romero, L., Mori, K. S., Morioka, T., Mori, S., Moro, S., Mortensen, B., Moskalevskyi, B., Muischnek, K., Murawaki, Y., Müürisep, K., Nainwani, P., Navarro Horñiacek, J. I., Nedoluzhko, A., Nešpore-Bērzkalne, G., Nguy˜ên Thị, L., Nguy˜ên Thị Minh, H., Nikaido, Y., Nikolaev, V., Nitisaroj, R., Nurmi, H., Ojala, S., Olúòkun, A., Omura, M., Osenova, P., Östling, R., Øvrelid, L., Partanen, N., Pascual, E., Passarotti, M., Patejuk, A., Paulino-Passos, G., Peljak-Łapińska, A., Peng, S., Perez, C.-A., Perrier, G., Petrova, D., Petrov, S., Piitulainen, J., Pirinen, T. A., Pitler, E., Plank, B., Poibeau, T., Popel, M., Pretkalniņa, L., Prévost, S., Prokopidis, P., Przepiórkowski, A., Puolakainen, T., Pyysalo, S., Rääbis, A., Rademaker, A., Ramasamy, L., Rama, T., Ramisch, C., Ravishankar, V., Real, L., Reddy, S., Rehm, G., Rießler, M., Rimkutė, E., Rinaldi, L., Rituma, L., Rocha, L., Romanenko, M., Rosa, R., Rovati, D., Roșca, V., Rudina, O., Rueter, J., Sadde, S., Sagot, B., Saleh, S., Salomoni, A., Samardžić, T., Samson, S., Sanguinetti, M., Särg, D., Saulīte, B., Sawanakunanon, Y., Schneider, N., Schuster, S., Seddah, D., Seeker, W., Seraji, M., Shen, M., Shimada, A., Shirasu, H., Shohibussirri, M., Sichinava, D., Silveira, N., Simi, M., Simionescu, R., Simkó, K., Šimková, M., Simov, K., Smith, A., Soares-Bastos, I., Spadine, C., Stella, A., Straka, M., Strnadová, J., Suhr, A., Sulubacak, U., Suzuki, S., Szántó, Z., Taji, D., Takahashi, Y., Tamburini, F., Tanaka, T., Tellier, I., Thomas, G., Torga, L., Trosterud, T., Trukhina, A., Tsarfaty, R., Tyers, F., Uematsu, S., Urešová, Z., Uria, L., Uszkoreit, H., Vajjala, S., van Niekerk, D., van Noord, G., Varga, V., Villemonte de la Clergerie, E., Vincze, V., Wallin, L., Walsh, A., Wang, J. X., Washington, J. N., Wendt, M., Williams, S., Wirén, M., Wittern, C., Woldemariam, T., Wong, T.-s., Wróblewska, A., Yako, M., Yamazaki, N., Yan, C., Yasuoka, K., Yavrumyan, M. M., Yu, Z., Žabokrtský, Z., Zeldes, A., Zeman, D., Zhang, M., and Zhu, H. (2019). Universal dependencies 2.4. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University. Cited on pages 51 and 76.

Nivre, J., De Marneffe, M.-C., Ginter, F., Goldberg, Y., Hajic, J., Manning, C. D., McDonald, R., Petrov, S., Pyysalo, S., Silveira, N., et al. (2016a). Universal Dependencies v1: A Multilingual Treebank Collection. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, pages 1659–1666. Cited on page 9.

Nivre, J., de Marneffe, M.-C., Ginter, F., Goldberg, Y., Hajič, J., Manning, C. D., McDonald, R., Petrov, S., Pyysalo, S., Silveira, N., Tsarfaty, R., and Zeman, D. (2016b). Universal Dependencies v1: A Multilingual Treebank Collection. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 1659–1666, Portorož, Slovenia. European Language Resources Association (ELRA). Cited on pages 10 and 25.

Nivre, J. and Nilsson, J. (2005). Pseudo-Projective Dependency Parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 99–106, Ann Arbor, Michigan. Association for Computational Linguistics. Cited on pages 14, 17, and 46.

Oepen, S., Kuhlmann, M., Miyao, Y., Zeman, D., Flickinger, D., Hajič, J., Ivanova, A., and Zhang, Y. (2014). SemEval 2014 task 8: Broad-coverage semantic dependency parsing. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 63–72, Dublin, Ireland. Association for Computational Linguistics. Cited on page 107.

Palmer, M., Gildea, D., and Kingsbury, P. (2005). The Proposition Bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31(1):71–106. Cited on pages 27 and 91.

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). BLEU: a Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics. Cited on pages 39, 53, and 99.

Puduppully, R., Zhang, Y., and Shrivastava, M. (2016). Transition-based Syntactic Linearization with Lookahead Features. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 488–493. Cited on page 17.

Qi, P., Zhang, Y., Zhang, Y., Bolton, J., and Manning, C. D. (2020). Stanza: A python natural language processing toolkit for many human languages. In *Proceedings of the 58th*

Annual Meeting of the Association for Computational Linguistics: System Demonstrations, pages 101–108, Online. Association for Computational Linguistics. Cited on page 89.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386. Cited on page 18.

Schmaltz, A., Rush, A. M., and Shieber, S. (2016). Word Ordering Without Syntax. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2319–2324, Austin, Texas. Association for Computational Linguistics. Cited on page 16.

Sgall, P., Hajičová, E., and Panevová, J. (1986). *The Meaning of the Sentence in Its Semantic and Pragmatic Aspects*. Dordrecht:Reidel Publishing Company and Prague:Academia. Cited on page 9.

Straka, M. and Straková, J. (2017). Tokenizing, POS Tagging, Lemmatizing and Parsing UD 2.0 with UDPipe. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 88–99, Vancouver, Canada. Association for Computational Linguistics. Cited on page 97.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. *arXiv preprint arXiv:1409.3215*. Cited on page 21.

Tai, K. S., Socher, R., and Manning, C. D. (2015). 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)*, volume 1, pages 1556–1566. Cited on pages 23 and 31.

Temperley, D. (2007). Minimization of dependency length in written english. *Cognition*, 105(2):300–333. Cited on page 2.

Temperley, D. and Gildea, D. (2018). Minimizing syntactic dependency lengths: typological/cognitive universal? *Annual Review of Linguistics*, 4:67–80. Cited on page 25.

Tesnière, L. (1959). *Eléments de syntaxe structurale*. Paris, Klincksieck. Cited on pages 1 and 9.

Tesnière, L. (2015). *Elements of structural syntax*. John Benjamins Publishing Company. Cited on pages 1 and 9.

Tillmann, C. and Ney, H. (2003). Word Reordering and a Dynamic Programming Beam Search Algorithm for Statistical Machine Translation. *Computational Linguistics*, 29(1):97–133. Cited on page 16.

Tily, H. (2010). *The role of processing complexity in word order variation and change*. Stanford University. Cited on page 66.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *NIPS*. Cited on pages 22 and 108.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. (2017). Graph attention networks. *arXiv preprint arXiv:1710.10903*. Cited on page 24.

Vinyals, O., Bengio, S., and Kudlur, M. (2015a). Order Matters: Sequence to Sequence for Sets. *arXiv preprint arXiv:1511.06391*. Cited on page 45.

Vinyals, O., Fortunato, M., and Jaitly, N. (2015b). Pointer Networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700. Cited on pages 22 and 35.

Voudouris, C. and Tsang, E. (1999). Guided Local Search and its Application to the Traveling Salesman Problem. *European journal of operational research*, 113(2):469–499. Cited on page 42.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Philip, S. Y. (2020). A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*. Cited on page 24.

Yu, X., Falenska, A., Haid, M., Vu, N. T., and Kuhn, J. (2019a). IMSurReal: IMS at the Surface Realization Shared Task 2019. In *Proceedings of the 2nd Workshop on Multilingual Surface Realisation (MSR 2019)*, pages 50–58, Hong Kong, China. Association for Computational Linguistics.

Yu, X., Falenska, A., and Kuhn, J. (2019b). Dependency length minimization vs. word order constraints: An empirical study on 55 treebanks. In *Proceedings of the First Workshop on Quantitative Syntax (Quasy, SyntaxFest 2019)*, pages 89–97, Paris, France. Association for Computational Linguistics. Cited on page 62.

Yu, X., Falenska, A., and Vu, N. T. (2017). A general-purpose tagger with convolutional neural networks. In *Proceedings of the First Workshop on Subword and Character Level Models in NLP*, pages 124–129, Copenhagen, Denmark. Association for Computational Linguistics.

Yu, X., Falenska, A., Vu, N. T., and Kuhn, J. (2019c). Head-First Linearization with Tree-Structured Representation. In *Proceedings of the 12th International Conference on Natural Language Generation*, pages 279–289, Tokyo, Japan. Association for Computational Linguistics.

Yu, X., Tannert, S., Vu, N. T., and Kuhn, J. (2020a). Fast and accurate non-projective dependency tree linearization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1451–1462, Online. Association for Computational Linguistics.

Yu, X., Tannert, S., Vu, N. T., and Kuhn, J. (2020b). IMSurReal too: IMS in the surface realization shared task 2020. In *Proceedings of the Third Workshop on Multilingual Surface Realisation*, pages 35–41, Barcelona, Spain (Online). Association for Computational Linguistics.

Yu, X. and Vu, N. T. (2017). Character composition model with convolutional neural networks for dependency parsing on morphologically rich languages. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 672–678, Vancouver, Canada. Association for Computational Linguistics.

Yu, X., Vu, N. T., and Kuhn, J. (2018). Approximate dynamic oracle for dependency parsing with reinforcement learning. In *Proceedings of the Second Workshop on Universal Dependencies (UDW 2018)*, pages 183–191, Brussels, Belgium. Association for Computational Linguistics.

Yu, X., Vu, N. T., and Kuhn, J. (2019d). Learning the Dyck language with attention-based Seq2Seq models. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 138–146, Florence, Italy. Association for Computational Linguistics. Cited on page 31.

Yu, X., Vu, N. T., and Kuhn, J. (2020c). Ensemble self-training for low-resource languages: Grapheme-to-phoneme conversion and morphological inflection. In *Proceedings of the 17th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 70–78, Online. Association for Computational Linguistics.

Zaslavskiy, M., Dymetman, M., and Cancedda, N. (2009). Phrase-Based Statistical Machine Translation as a Traveling Salesman Problem. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 333–341, Suntec, Singapore. Association for Computational Linguistics. Cited on page 16.

Zeman, D., Hajič, J., Popel, M., Potthast, M., Straka, M., Ginter, F., Nivre, J., and Petrov, S. (2018). CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–21, Brussels, Belgium. Association for Computational Linguistics. Cited on page 89.

Zhou, Y., Liu, C., and Pan, Y. (2016). Modelling Sentence Pairs with Tree-structured Attentive Encoder. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 2912–2922. Cited on page 23.

Zubizarreta, M. L. (1998). Prosody, focus, and word order. Cited on page 12.