SUPERVISED TRAINING ON SYNTHETIC LANGUAGES: A NOVEL FRAMEWORK FOR UNSUPERVISED PARSING

by

Dingquan Wang

A dissertation submitted to The Johns Hopkins University

in conformity with the requirements for the degree of

Doctor of Philosophy

Baltimore, Maryland

October, 2019

Abstract

This thesis focuses on unsupervised dependency parsing—parsing sentences of a language into dependency trees without accessing the training data of that language. Different from most prior work that uses unsupervised learning to estimate the parsing parameters, we estimate the parameters by *supervised* training on synthetic languages. Our parsing framework has three major components: *Synthetic language generation* gives a rich set of training languages by mix-and-match over the real languages; *surface-form feature extraction* maps an unparsed corpus of a language into a fixed-length vector as the syntactic signature of that language; and, finally, *language-agnostic parsing* incorporates the syntactic signature during parsing so that the decision on each word token is reliant upon the general syntax of the target language.

The fundamental question we are trying to answer is whether some useful information about the syntax of a language could be inferred from its surface-form evidence (*unparsed* corpus). This is the same question that has been implicitly asked by previous papers on unsupervised parsing, which only assumes an unparsed corpus to be available for the target language. We show that, indeed, useful features of the target language can be extracted automatically from an unparsed corpus, which consists only of gold part-of-speech (POS) sequences. Providing these features to our neural parser enables it to parse sequences like those in the corpus. Strikingly, our system has no supervision in the target language. Rather, it is a multilingual system that is trained end-to-end on a variety of *other* languages, so it learns a feature extractor that works well.

This thesis contains several large-scale experiments requiring hundreds of thousands of CPU-hours. To our knowledge, this is the largest study of unsupervised parsing yet attempted. We show experimentally across multiple languages: (1) Features computed from the unparsed corpus improve parsing accuracy. (2) Including thousands of synthetic languages in the training yields further improvement. (3) Despite being computed from unparsed corpora, our learned task-specific features beat previous works' interpretable typological features that require parsed corpora or expert categorization of the language.

Thesis Committee

Primary Readers

Jason Eisner (Primary Advisor)
Professor
Department of Computer Science
Johns Hopkins University

Tal Linzen
Assistant Professor
Department of Cognitive Science and Computer Science
Johns Hopkins University

Joakim Nivre
Professor of Computational Linguistics
Department of Linguistics and Philology
Uppsala University

Slav Petrov
Principal Scientist / Research Director
Google AI

Matt Post
Research Scientist
Human Language Technology Center of Excellence
Department of Computer Science
Johns Hopkins University

For Y.Z.

Acknowledgments

Portions of this thesis include four peer-reviewed papers (Wang and Eisner, 2016; Wang and Eisner, 2017; Wang and Eisner, 2018a; Wang and Eisner, 2018b). I would like to thank my adviser Jason Eisner, who contributed substantial edits on all of these papers as well as the rest chapters in this thesis (expecially Chapter 2). Jason is the most incredible advisor I have ever worked with. He is known for his long technical emails to collaborators. But when I first got one of his legendary emails, I was still blown away by its depth and vision.[1] When I was writing a paper in 2018, I still went back to his emails from 2014 for reference. Jason's extraordinarily high standards on papers make him the worst enemy *before* submission deadlines, and the best ally *after* submission deadlines, when it is time to improve the written and talk presentations further. His attitude towards research goes beyond the research itself, rather, he views our research as a means to serve the community, and I would like to emulated him in the future.

In addition to Jason, I'm grateful to the people who have served as committee members who went beyond the call of duty to support me on my journey toward this degree. Thanks to my Graduate Board Oral (GBO) exam committee, consisting of Kevin Duh, Sanjeev Khudanpur, Benjamin Van Durme, Gèraldine Legendre (alternate),

[1]Also the length—I printed out my first one, which took 15 A4 pages.

Colin Wilson, and David Yarowsky (alternate), for their insightful comments on the proposal. Thanks to my dissertation committee, consisting of Tal Linzen, Joakim Nivre, Slav Petrov, and Matt Post, for their patience in reading through this document and suggesting improvements to hold this work to a higher standard.

Most of this work is funded by the U.S. National Science Foundation under Grant No. 1423276 and 1718846. The state of Maryland provides indispensable computing resources via the Maryland Advanced Research Computing Center (MARCC), which makes the massive experiments possible. Early discussions and code prototypes of Wang and Eisner (2016) are provided by Raman Arora, Matt Gormley and Sharon Li. Lillian Lee, who serves as the co-editor-in-chief of the TACL, gave careful corrections on Wang and Eisner (2016), Wang and Eisner (2017), and Wang and Eisner (2018a). Denise Link-Farajali and Anne Colgan from the Center for Leadership Education gave thorough and detailed writing suggestions, where Denise helped on the abstract, Chapters 1 and 2; and Anne helped on the acknowledgments and Chapter 7. Outside the JHU, this work benefited from productive discussions with Regina Barzilay, Emily Bender, Michael Collins, Adam Fisch, Jiang Guo, Mitch Marcus, Graham Neubig, Mohammad Sadegh Rasooli, Lyle Ungar, Wenpeng Yin, and Mo Yu.

Of course, as an Argonaut,[2] I'm fortunate to have worked closely with other Argonauts, including Nicholas Andrews, Jacob Buckman, Ryan Cotterell, Nathaniel (Wes) Filardo, Matthew Francis-Landau, Juneki Hong, Xiang (Lisa) Li, Xiaochen Li, Chu-Cheng Lin, Chenxi Liu, Becky Marvin, Hongyuan Mei, Sebastian Mielke, Guanghui Qin, Pushpendre Rastogi, Nanyun (Violet) Peng, Adi Renduchintala, Darcey Riley, Tim Vieira, Shijie Wu, Akshay Srivatsan, Adam Teichert, and Mozhi (Miles)

[2]`https://www.cs.jhu.edu/~jason/Argo/`

Zhang, who are incredibly smart problem solvers and generous helpers. I want to thank to them for brainstorming ideas, and giving feedback on paper drafts and practice talks.

Beyond Argo, I'm a proud member of the CLSP and CS community at JHU and grateful to my colleagues there for fostering a relaxed, creative and collaborative atmosphere. They are: Tongfei Chen, Shuoyang Ding, Seth Ebner, Dongji Gao, Lv Hang, Huda Khayrallah, Rebecca Knowles, Keith Levin, Ke Li, Chunxi Liu, Xutai Ma, Arya McCarthy, Poorya Mianjy, Adam Poliak, Pamela Shapiro, Suzanna Sia, Shuo Sun, Yiming Wang, Zachary Wood-Doughty, Winston Wu, Patrick Xia, Hainan Xu, Sheng Zhang, and Xiaohui Zhang. Thanks to Zachary Burwell, Ruth Scally, and Cathy Thornton for managing the administrative stuff so smoothly that it freed me from the distractions outside the research.

I thank my mom and dad for their endless and unconditional support so that I could pursue the degree this far. I owe you so much especially for not being able to come back and spend Chinese New Years with you since 2013.

Finally, thanks to Yuehan, who is the motivator, the encourager, the morale booster, the problem solver, the complaint listeneer, the happiness sharer, the best friend, the love of my life, and the true author of this work.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Unsupervised learning aims to discover deep knowledge from surface-form data, which has been a longstanding research question in the machine learning community. In this thesis, we are working on one typical unsupervised learning problem in the realm of natural language processing (NLP)—unsupervised parsing. Unsupervised parsing tries to parse sentences of a language into trees by *only* accessing unparsed data of that language, where the most important step is inferring parsing parameters from unparsed data. This raises a more fundamental question that whether some useful information about the syntax of a language could be inferred from its surface evidence, which is the *main* research question this thesis is set to answer. In this opening chapter, we carefully spell out our motivation by exploring three topics: 1) The importance of unsupervised parsing, 2) main challenges of this task, and 3) the intuition of our approach in response to these challenges.

Beyond this chapter, the rest of this thesis is organized as follows: Chapter 2 will formally introduce our framework by connecting to the unsupervised learning literature, and, therefore, states the challenge of data sparsity. Chapter 3 delves into this challenge and proposes to enrich the data by synthesizing "new languages" via a mix-and-match procedure among the real languages. The two chapters following Chapter 3 will apply our framework to two unsupervised learning tasks: Chapter 4 focuses on predicting the property of each dependency relation *type*, which we call the *fine-grained syntactic typology*; and Chapter 5 is about unsupervised dependency parsing, which is our main task. Finally, Chapter 6 will introduce a novel idea departing from Chapter 3, where we generate synthetic languages that "look like" the target language.

1.1 Parse Trees in the Era of Neural Networks

The output of unsupervised parsing (as well as supervised parsing) is a parse tree for each sentence. Before this era of neural networks, the parse tree served as the main device in encoding the *meaning* of a sentence. A number of tasks benefit from parse trees as an intermediate representation. For example, Chiang (2005) used constituency trees to control the granularity of reordering when translating between two languages with different word orders. Mihalcea and Tarau (2004) applied the PageRank (Brin and Page, 1998) algorithm over the dependency graph to rank words for keyword extraction. Gildea and Palmer (2002) found parse trees are more effective than flat chunk sequences for semantic rule identification. Culotta and Sorensen (2004) defined a metric over the tree space to use kernel-based support vector machines (Collins and Duffy, 2001) for relation classification. This kernel-based approach was also applied to sentiment analysis (Agarwal et al., 2011). In the realm of question answering (QA), parsing has been a standard step in transforming sentences into logical forms that machines could interpret (Zettlemoyer and Collins, 2007).

A weakness of the tree-based approach is that the quality of parses heavily bottlenecks the performance. In other words, the error in the parses will cascade to those downstream tasks. Thus, researchers are seeking alternatives that directly model the sentences, which becomes more favorable due to the recent advances in neural networks for NLP. One notable example is the success of the neural sequence to sequence framework (Sutskever, Vinyals, and Le, 2014; Cho et al., 2014a) that extracts sentence representations through some encoding architectures whose parameters are tuned through end-to-end training. Typical encoding architectures include recurrent neural networks (Sutskever, Vinyals, and Le, 2014; Cho et al., 2014a; Radford et al., 2019), attention-based models (Bahdanau, Cho, and Bengio, 2015), stack-layered models (Peters et al., 2018), and hybrid models such as the Transformer (Vaswani et al., 2017; Devlin et al., 2019). In addition to eliminating the intermediate steps, the success of neural networks relies on two other factors: 1) It is expressive enough that doesn't require expensive feature engineering, and 2) its end-to-end nature of facilitates pre-training and fine-tuning, where trained models on cloze (Devlin et al., 2019), language modeling (Peters et al., 2018), and translation (McCann et al., 2017) have shown to be beneficial as initialization for other tasks and gain improvements by further tuning on those task-specific objectives.

As a result, whether parsing is still useful becomes the question. The following discussion will demonstrate that the answer is yes.

1.1.1 Making neural models linguistically informed

While neural networks have been the backbone of most modern NLP systems, additional linguistic information such as parse trees has proven to be beneficial. The recursive neural network (RecNN) (Goller and Kuchler, 1996; Socher, Manning, and Ng, 2010) represents just such an example that explicitly uses parse trees to build up sentence representations. Compared to recurrent neural networks (RNNs) that build up sentence meaning through time, RecNN is a linguistically principled approach that corresponds better to compositional semantics. On the other hand, RecNN is still a neural network model where the composition at each tree node is formed by neural gates that take vector representations as inputs. As the result, RecNN has taken its place in the era of neural networks and has been actively studied. Early work by Socher et al. (2013) demonstrates its effectiveness on sentiment classification, where they employed constituency trees with the re-annotation of sentiment polarity at each node. This architecture was later improved by borrowing the combining mechanism from Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997). Bowman, Potts, and Manning (2015) examined RecNN-based meaning representations over a range of natural language inference tasks and verified their effectiveness. From a broader view, the popularity of RecNN reflects an increasing interest in representation learning on structures. In addition to modeling sentiment, the RecNN-style approach has been shown to be useful for semantic role labeling (Marcheggiani and Titov, 2017) and word embedding induction (Levy and Goldberg, 2014). Moreover, techniques have been generalized from trees to graph structures (Hamilton et al., 2018; Xu et al., 2019), which has a wider range of applications in recommender systems, knowledge graphs, and social networks. We won't delve deeply into this topic in that it is less relevant to this thesis; nonetheless, we have good faith in the future uses of parse trees.

As well as directly enforcing the tree structures into the network architecture, an alternative is leveraging them as soft constraints. Roth and Lapata (2016) enrich token representations by running RecNN over dependency trees in addition to the LSTM features and gain state-of-the-art performance on semantic role labeling. Strubell et al. (2018) later push the frontier by jointly training on POS tagging and dependency parsing through multitask learning.

1.1.2 Understanding neural models

Despite the success in NLP tasks, neural networks are commonly acknowledged to have low interpretability. Thus, in parallel to novel neural architectures, research has been conducted to explain their underlying mechanism. A usual approach in this vein is evaluating neural models on predicting some linguistic properties as *probing* tasks, where parse trees are commonly used. Qian, Qiu, and Huang (2016) used the LSTM vectors (cell, gate and output values) at each word to predict its depth in a dependency tree, but didn't find a definite correlation. Conneau et al. (2018) studied the sentence representation produced by LSTMs and found it capable of predicting the depth of the entire tree. Linzen, Dupoux, and Goldberg (2016) and Ravfogel, Goldberg, and Linzen (2019) discovered that LSTMs are able to determine subject-verb agreement, where they used an English treebank and its synthetic variations. Similar findings (Lin, Tan, and Frank, 2019) later extend to the Transformer architecture (Devlin et al., 2019).

1.1.3 Guiding model transfer across domains

Model transfer, or transfer learning, is a useful technique for applying NLP systems (trained on rich-resource domains) to low-resource domains which doesn't have enough training examples. As most tree schemes are based on some domain-independent representations, they can serve as the media to carry information during transfer. In machine translation, researchers (Chiang, 2005; Chiang, 2007; Huang, Čmejrek, and Zhou, 2010) used parse trees generated from a synchronous context-free grammar to model parallel sentences, where tree nodes on both sides shared the same vocabulary of the syntactic categories. Moreover, this shared syntactic representation has been shown to be useful for guiding the induction of bilingual word embeddings (Duong et al., 2015a). The potential of parse trees for model transfer has motivated a recent interest in building multi-domain treebanks using unified syntactic vocabulary (McDonald et al., 2013; Nivre et al., 2016). The most well-known project is the Universal Dependencies (Nivre et al., 2016)—a set of dependency treebanks that covers multiple languages and genres.

1.2 Dependency Structure

In this thesis, we consider parsing sentences into dependency trees. As shown in Figure 1.1, the dependency tree of a sentence is a directed graph of labeled binary syntactic relations between words. The linguistic foundation of dependency structure derives from a diverse set of grammar formalisms (Tesnière, 1959; Hudson, 1984; Hellwig, 1986; Sgall et al., 1986; Mel'cuk, 1988) that share a central concept called *dependency*.[1] A dependency between two words could be represented by an "arrow" from one word called *head* to the other called *dependent*. In linguistics, this dependency reflects grammatical function, where the dependent is the modifier or

[1]Nivre (2005) gives a nice survey on the literature of dependency grammar.

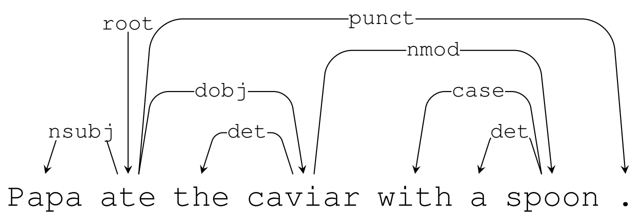


Figure 1.1: An English dependency tree in the UD version 1 scheme.

complement of the other. A variety of criteria (Zwicky, 1985; Hudson, 1984) determine the head; e.g., in the Stanford Dependencies (SD) scheme (Marneffe et al., 2014), the head of a prepositional phrase (PP) is the preposition, which is the *dependent* in the Universal Dependency (UD) scheme (Nivre et al., 2015).

1.2.1 Reason for using dependency structure

Our primary rationale for choosing dependency structure is rather practical—we want our system to generalize well across many languages. Thus, the consideration is more about the coverage of languages than formalism. Dependency structure is an ideal candidate as it has treebanks over a wide range of languages, owing to the contributors of the UD project (Nivre et al., 2016), who have thus far annotated dependency treebanks for 83 languages (Nivre et al., 2019). On the other hand, as dependency structure is usually flatter than X-bar structure, it allows more reorderings for synthetic data generation, which is a crucial technique in this thesis (Chapters 3 and 6).

Indeed, dependency formalism is favorable to NLP literature. Therefore, it is worth commenting on this popularity in general, where we compare it to constituency structure—another popular syntactic formalism. For historical reasons, partially because of the seminal work of Chomsky (1957) and the early attention to English data, the constituency formalism has been the dominating syntactic theory applied

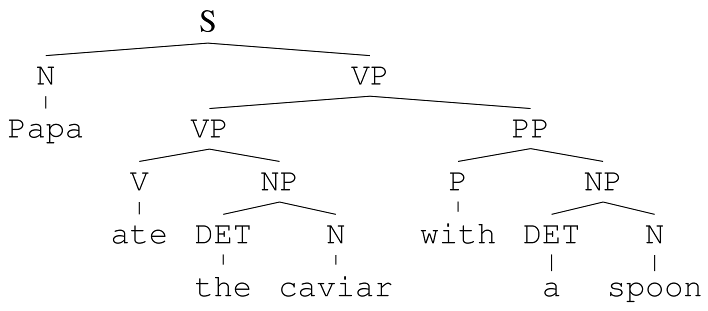


Figure 1.2: An English constituency tree on the same sentence as Figure 1.1.

to NLP research. To bring this to light, the influential Penn Treebank (Marcus, Marcinkiewicz, and Santorini, 1993) is annotated by constituency trees. As shown in Figure 1.2, one obvious distinction of a constituency tree from a dependency tree (Figure 1.1) is that it characterizes the relationships among phrases, yet the latter directly characterizes relationships among words. We depart from this difference and summarize the advantages of the dependency formalism for the NLP community.

Dependency structure is minimal in that it has fewer nodes than the constituency structure (Maxwell, 2013). This simplicity stands out from a computational perspective, where it is more efficient for parsing. Thus, a majority of the recent parsing techniques (Nivre, 2003; McDonald et al., 2005; Chen and Manning, 2014; Dyer et al., 2015; Kiperwasser and Goldberg, 2016) are developed for dependency structures, which results in a rich and rapidly growing inventory of dependency parsers. On the other hand, the simplified representation eases the process of annotating and interpreting the syntactic structure for people with less linguistic expertise.

A dependency structure gives an explicit description of the predicate-argument relationships in the sentence, which has been widely used for modeling shallow meaning representations. In contrast, a constituency structure requires augmentation for this information (De Marneffe, 2012). Also, the link structure is suitable for

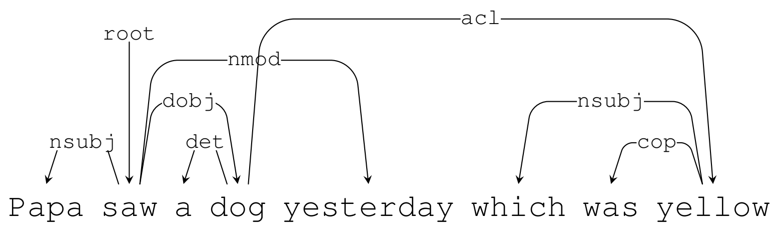


Figure 1.3: A non-projective dependency tree on a sentence that has discontinuity phenomenon.

characterizing the discontinuity phenomenon that is usually caused by syntactic movements. Figure 1.3 gives such an example, which is called a *non-projective* dependency tree. As we can see, it is difficult to annotate this sentence with a constituency tree because the phrase structure is broken. Although the discontinuity is rare in English, it is common for other languages that have free word orders. As a result, the dependency formalism is more adaptable to the vast diversity of human languages.

As a disclaimer, discussion thus far only rationalizes the popularity of dependency structure from a practical point of view, but doesn't conclude its supremacy over constituency structure in general. Each of the two formalisms has a deep root in the literature of linguistics (Chomsky, 1957; Tesnière, 1959).[2] The argument over the choice between them is not new (Mel'cuk, 1988; Kahane, 2012), where the debates mainly center around the correspondence to the human mind and its expressive power. On the other hand, work also exists on their equivalence (only for projective dependency trees) that proposes some rule-based (Collins, 2003; Eisner and Satta, 1999; Eisner, 2000) or learning-based (Kong, Rush, and Smith, 2015) methods to transform one to another. In that the debate is less relevant to this thesis, we will make

[2]The notion of dependency grammar may even be traced back to Pāṇini's grammar of Sanskrit centuries before the Common Era (Kruijff, 2002; Nivre, 2005).

no further attempt at reviewing this literature.

1.3 Unsupervised Dependency Parsing

When referring to dependency parsing, one usually refers to *supervised* dependency parsing—parsers that are trained on a treebank of known parses in the target language. The study of the supervised approach dominates the dependency parsing research, and its progress has been so rapid (Nivre, 2003; McDonald et al., 2005; McDonald, 2006; Nivre, 2008; Chen and Manning, 2014; Dyer et al., 2015; Kiperwasser and Goldberg, 2016; Dozat and Manning, 2017; Andor et al., 2016; Ma et al., 2018; Fernández-González and Gómez-Rodríguez, 2019) that some supervised dependency parsers have even approached human-level performance[3] on some datasets due to the boom of neural networks. In contrast, the progress of *unsupervised* dependency parsers has been slow, and they have apparently not been used in any downstream NLP systems (Mareček, 2016). An unsupervised parser does not have access to a treebank, but only to a corpus of unparsed sentences in the target language. This is the main task of this thesis.

The research on unsupervised dependency parsing origins from a general interest in inducing deep structures from surface form representations. Most of the previous work is on the induction of generative rules from a set of surface strings, which is a task referred to as *grammar induction* (Fu and Booth, 1975; Angluin and Smith, 1983; Carroll and Charniak, 1992). The notion of generative rules is of interest to communities beyond NLP, such as programming languages (Smith, 1982), bioinformatics (Sakakibara et al., 1994), and computer vision (Zhu and Mumford, 2007; Tu, 2015).

[3]In the announcement of the SyntaxNet (Andor et al., 2016) project, they found the agreement score of trained linguists on the Penn Treebank is 96-97%, where parsers achieved 95%+ since 2016.

In the literature of linguistics, grammar induction is driven by the curiosity about the human language acquisition process, especially to Chomsky's well-known Universal Grammar (UG) theory (Chomsky, 1965; Chomsky, 1981; Chomsky and Lasnik, 1993). The UG theory claims that human babies must use an innate estimator that is statistically efficient on real human languages, enabling them to acquire language accurately from a relatively small number of unsupervised examples, which is not possible for domain-independent algorithms. This statement is famously referred to as *the Poverty of the Stimulus*. In order to support or reject the UG theory, one direct approach is to just find such a domain-independent algorithm (Clark, 2001), which motivates grammar induction research. Admittedly, researchers have acknowledged the weakness of the connection between grammar induction and UG, since the setup of grammar induction is far from a real language learning device (Klein and Manning, 2005). On the one hand, grammar induction is easier because it usually takes some grammatical sentences as input and generates a set of probabilistic rules. In contrast, human babies are exposed to a complex environment that has numerous incomplete and noisy utterances as input, and the output is the linguistic knowledge stored in their brains. On the other hand, grammar induction is more difficult because babies have access to a much richer and more interactive environment than pure grammatical input.

For practical NLP applications, unsupervised parsing is useful for reducing the human annotation cost. As the fuel of data-driven models, the amount and quality of labeled data have become increasingly important. For dependency parsing, the necessity of annotating new data is longstanding as a new domain or language always comes onto the horizon. A successful unsupervised parser may lift the burden of human annotation. Even if it has mediocre performance, the output may serve as a starting point to reduce further manual efforts.

Throughout the decades, the most common unsupervised parsing approach has been rule-based grammar induction. The basic idea is first inducing an explicit probabilistic context-free grammar (PCFG) from the unparsed corpus by maximum likelihood estimation and then using that PCFG to parse sentences by CKY algorithms. This approach has encountered two major challenges:

- **Search error:** Most formulations of grammar induction involve optimizing a highly non-convex objective function such as likelihood. The optimization is typically NP-hard (Cohen and Smith, 2012), and approximate local search methods tend to get stuck in local optima.

- **Model error:** Likelihood does not correlate well with parsing accuracy anyway (Smith, 2006, Figure 3.2). Likelihood optimization seeks latent trees that help to predict the observed sentences, but these unsupervised trees may use a non-standard syntactic analysis or even be optimized to predict non-syntactic properties such as topic. We seek a *standard syntactic analysis*—what Smith (2006) calls the MATCHLINGUIST task.

Facing these challenges, study on unsupervised parsing has switched attention to more practical directions that are less interested in deriving formal grammars than directly obtaining parsers. One notable trend is cross-lingual transfer, which assumes treebanks for some resource-rich languages to be available and transfers the trained parser on those languages to the target languages. For a detailed literature review, see Section 5.2.

1.4 Our Approach: An Artificial Linguist

In this thesis, we address both challenges above with a *supervised learning framework*, whose objective function is easier to optimize and explicitly tries to match linguists' standard syntactic analyses. Our inspiration comes from an intuition that this task—like others that engineers, linguists, or human learners might face—may be solvable with general knowledge about the distribution of human languages. An experienced linguist can sometimes puzzle out the structure of a new language. The reader may be willing to guess a parse for the gold POS sequence `VERB DET NOUN ADJ DET NOUN`. After all, adjectives usually attach to nouns (Naseem et al., 2010), and the adjective in this example seems to attach to the first noun—not to the second, insomuch as determiners usually fall at the edge of a noun phrase. Meanwhile, the sequence's sole verb is apparently followed by two noun phrases, which suggests either VSO (verb-subject-object) or VOS order—and VSO is a good guess as it is more common (Dryer and Haspelmath, 2013). Observing a corpus of additional POS sequences might help resolve the question of whether this language is primarily VSO or VOS, for example, by guessing that short noun phrases in the corpus (for example, unmodified pronouns) are more often subjects.

Thus, we propose to solve the task by training a kind of *artificial linguist* that can do such analysis on corpora of new languages. This is a general approach to developing an unsupervised method for a specific type of dataset: tune its structure and hyperparameters so that it works well on actual datasets *of that sort*, and then apply it to *new* datasets. In this thesis, this specific type of dataset refers to a dependency treebank.

For example, consider clustering—the canonical unsupervised problem. What constitutes a useful cluster depends on the type of data and the application. Basu, Jacobs, and Vanderwende (2013) developed a text clustering system specifically to aid teachers. Their "Powergrading" system can group all the student-written answers to a *novel* question, having been trained on human judgments of answer similarity for *other* questions. Their novel questions are analogous to our novel languages: their unsupervised system is specifically tailored to match teachers' semantic similarity judgments within any corpus of student answers, just as ours is tailored to match linguists' syntactic judgments within any corpus of human-language POS sequences. Other NLP work on supervised tuning of unsupervised learners includes strapping (Eisner and Karakos, 2005; Karakos et al., 2007), which tunes with the help of both real and synthetic datasets, just as we will (Chapter 3).

Are such systems really "unsupervised"? Yes, in the sense that they are able to discover desirable structure in a *new* dataset. Unsupervised learners are normally crafted using assumptions about the data domain. Their structure and hyperparameters may have been manually tuned to produce pleasing results for typical datasets in that domain. In the domain of POS corpora, we simply scale up this practice to *automatically* tune a large set of parameters, which later guide our system's search for linguist-approved structure on each new human-language dataset. Our system should be regarded as "supervised" if the examples are taken to be entire languages: after all, we train it to map unlabeled corpora to usefully labeled corpora. But once trained, it is "unsupervised" if the examples are taken to be the sentences within a given corpus: by analyzing the corpus, our system figures out how to map sentences of *that* language to parses, without any labeled examples in that language.

1.4.1 The importance of the synthetic training languages

We hope for our system to do well, on average, at matching real linguist-parsed corpora of real human languages. We therefore tune its parameters $\mathbf{\Theta}$ on such treebanks. UD provides training examples *actually* drawn from that distribution $\mathcal{D}$ over treebanks—but alas, rather few. Thus, to better estimate the expected performance of $\mathbf{\Theta}$ under $\mathcal{D}$, we augment our training data with *synthetic* treebanks, which will be introduced in Chapter 3.

Ideally we would have sampled these synthetic treebanks from a careful estimate $\hat{\mathcal{D}}$ of $\mathcal{D}$: for example, the mean of a Bayesian posterior for $\mathcal{D}$, derived from prior assumptions and UD evidence. However, such adventurous "extrapolation" of unseen languages would have required actually constructing such an estimate $\hat{\mathcal{D}}$—which would embody a distribution over semantic content and a full theory of universal grammar! The synthetic treebanks were derived more simply and more conservatively by "interpolation" among the actual UD corpora. They combine observed parse trees (which provide *attested* semantic content) with stochastic word order models trained on observed languages (which attempt to mimic *attested* patterns for presenting that content). The sampling distribution $\hat{\mathcal{D}}$ still offers moderately varied synthetic datasets, which remain moderately realistic, as they are limited to phenomena observed in UD.

As Chapters 3 and 6 will point out, synthetic examples have been used in many other supervised machine learning settings. A common technique is to exploit invariance: if real image $\mathbf{z}$ should be classified as a cat, then so should a *rotated* version of image $\mathbf{z}$. Our technique is the same! We assume that if real corpus $\boldsymbol{u}$ should be parsed as having certain dependencies among the word tokens, then so should a version of corpus $\boldsymbol{u}$ in which those tokens have been *systematically permuted* in a *linguistically* *plausible* way. This is analogous to how rotation systematically transforms the image (rotating all pixels through the *same* angle) in a physically plausible way (as real objects do rotate relative to the camera). This systematicity is needed to ensure that the task on synthetic data is feasible. In our case, the synthetic corpus then provides many sentences that have been similarly permuted, which may *jointly* provide enough clues to guess the word order of this synthetic language (for example, VSO vs. VOS in Section 1.4) and thus recover the dependencies. See Section 6.2 for related discussion.

With enough good synthetic languages to use for training, even nearest-neighbor could be an effective method. That is, one could obtain the parser for a test corpus simply by copying the trained parser for the most similar training corpus (under some metric). Section 3.7.1 explores this approach of "single-source transfer" from synthetic languages. Yet with only thousands of synthetic languages, perhaps no *single* training corpus is sufficiently similar.[4] To draw on patterns in *many* training corpora to figure out how to parse the test corpus, we will train a single parser that can handle all of the training corpora (Ammar et al., 2016).

1.5 Key Limitation

Most of our experiments assume that *gold* POS tags of the target languages are available, which requires extra supervision in practice. In other words, we do not aim to build a practical system that should consider combining unsupervised techniques with yet other distantly supervised resources. But to develop those unsupervised techniques in the first place, we feel that it is a useful research strategy to study them in isolation to avoid confounds. Thus, we follow the setup of the most unsupervised

[4]Chapter 6 does investigate synthesis "on demand" of a permuted training corpus that is as similar as possible to the test corpus.

and cross-lingual transfer work (Naseem et al., 2010; McDonald, Petrov, and Hall, 2011; Täckström, McDonald, and Nivre, 2013; Zhang and Barzilay, 2015; Rosa and Žabokrtský, 2015a), which assumes an unparsed corpus that has gold part-of-speech (POS) sequences to be available for the language to be parsed. Under this assumption, as we will show that the answer is yes, and information can be extracted and used to obtain actual parses. Still, Sections 3.7.4, 4.6.4 and 5.6.6 depict experimental results on more realistic scenario where POS tags have noise.

Chapter 2

Formal Approach

In this chapter, we present our high-level idea. We start by formulating grammar induction as a statistical estimation problem. Given an observed dataset (corpus), we would like to identify the parameters (grammar) that gave rise to that dataset. In a Bayesian setting where we have a prior over the parameters, we could do this by performing maximum a posteriori (MAP) estimation—or more generally, by constructing a Bayes estimator. A Bayes estimator is an estimator that achieves minimum *expected* loss (for some given loss function).

These estimators are well-defined in a Bayesian setting. However, while principled, they tend to be computationally intractable, including in the settings of interest to us. We therefore propose *amortized* Bayesian estimation: *construct an estimator by training a supervised prediction method on samples from the Bayesian generative model*. This is a general (and apparently novel) proposal for practical Bayesian estimation. Rather than directly use the prior to *separately* analyze each dataset, we amortize the work of analysis by "compiling" the prior into an potentially fast estimator that is *generally* good at analyzing datasets that were generated from parameters drawn from the prior. The estimator should then be similarly good at analyzing real test datasets—granted our prior assumption that the test datasets were generated in precisely this way.

In our grammar induction setting, amortized Bayesian estimation would train a supervised system to accurately predict grammar from corpus on a set of synthetic languages drawn from the prior. For these samples, the grammar is known because it is drawn as part of the sampling process. We can then apply our trained estimator to predict grammar from corpus on real languages where the grammar is not known.

Our amortized approach can also be used for other forms of Bayesian inference. Estimation specifically tries to recover the parameter vector. In the more general case of inference, we wish to recover properties of the parameter vector (e.g., typology in Chapter 4) or other latent variables generated *en route* to the observations (e.g., parse trees in Chapter 5). Again, we can use samples from the prior to train a supervised predictor of these quantities.

In practice, it is difficult to write down an explicit Bayesian prior over natural languages. Fortunately, our amortized method does not actually need an explicit representation of the prior. It only needs a sample of languages from the prior. Thus, we could train our supervised predictor on a random sample of naturally occurring human languages—or a larger synthetic sample of "plausible" human languages as we will describe in Chapter 3. This sample "stands in" for the prior. The insight is that if we train our estimator to predict grammatical properties from the corpus on many plausible languages, with low average loss and low generalization error, then it should generalize to real languages.

2.1 Bayesian Estimation and Inference

We will use uppercase characters for random variables and their lowercase equivalents for the corresponding instances.

The general setting for Bayesian estimation and inference is a generative probability model of the form

$$p(\boldsymbol{G} = \boldsymbol{g}) \cdot p(\boldsymbol{Y} = \boldsymbol{y} \mid \boldsymbol{G} = \boldsymbol{g}) \cdot p(\boldsymbol{X} = \boldsymbol{x} \mid \boldsymbol{G} = \boldsymbol{g}, \boldsymbol{Y} = \boldsymbol{y}) \tag{2.1}$$

where $\boldsymbol{g}$ is a parameter vector, $\boldsymbol{y}$ is an optional latent variable, and $\boldsymbol{x}$ is the observed dataset. Given $\boldsymbol{x}$, estimation tries to recover $\boldsymbol{g}$, while inference tries to recover $\boldsymbol{y}$ or some other function of $\boldsymbol{g}$ and $\boldsymbol{y}$.

The methods of this chapter are applicable to any setting of the form in Equation (2.1). In the specific case of grammar induction,

- $\boldsymbol{g}$ is a probabilistic grammar in some family of possible grammars: it specifies a distribution over derivation trees. For example, $\boldsymbol{g}$ might be a PCFG (Section 1.3).
- $\boldsymbol{y}$ is a treebank of derivations drawn from that distribution. They are said to be "generated" by the grammar $\boldsymbol{g}$.
- $\boldsymbol{x}$ is the observed corpus, which is deterministically obtained as the yield strings of the trees in the treebank.[1] In other words, $p(\boldsymbol{X} = \boldsymbol{x} \mid \boldsymbol{G} = \boldsymbol{g}, \boldsymbol{Y} = \boldsymbol{y})$ is 1 if $\boldsymbol{x}$ is the fringe of $\boldsymbol{y}$, and is 0 otherwise.

[1]In Section 2.4.3 and chapter 5, we will also observe the yield strings $\boldsymbol{u}$ of some additional latent trees drawn from $\boldsymbol{g}$. The only difference between $\boldsymbol{x}$ and $\boldsymbol{u}$ is that in our amortized inference approach, our synthetic sample reveals the parses $\boldsymbol{y}$ for the sentences $\boldsymbol{x}$, but does not reveal the parses for $\boldsymbol{u}$. This is because some ways of constructing synthetic samples might include some unparsed text $\boldsymbol{u}$.

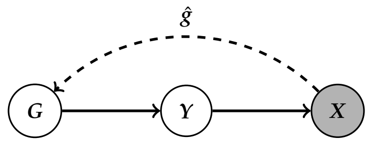


Figure 2.1: The setup of the grammar induction, where the shaded variable $\boldsymbol{X}$ is observed. Grammar induction wants to discover the *latent grammar* $\mathbf{G}$ from the observation $\boldsymbol{X}$ through an estimator $\hat{\boldsymbol{g}}$ in the dashed arrow.

In this setting, estimation of $\boldsymbol{g}$ corresponds to grammar induction, whereas inference of $\boldsymbol{y}$ corresponds to parsing (Chapter 5). Inferring aggregate syntactic properties of the language (Chapter 4) is also a form of inference.

As will be discussed in Section 2.4, one distinction between our approach and grammar induction is that the latter makes a strong assumption: $\boldsymbol{G}$ is a real-valued vector, which characterizes a *grammar*. As shown in Figure 2.1, grammar induction is a *backward* process to discover an *estimate* $\hat{\boldsymbol{g}}(\boldsymbol{x})$ from observed $\boldsymbol{x}$ in order to approximate the true $\boldsymbol{g}$. Function $\hat{\boldsymbol{g}}$ is called an *estimator*, and the estimate $\hat{\boldsymbol{g}}(\boldsymbol{x})$ is used to parse any sentences in this language. Many types of estimator $\hat{\boldsymbol{g}}$ have been proposed in the literature of grammar induction, and we will introduce a novel one (Section 2.2) on the way to our final approach (Section 2.4). Before that, Section 2.1.1–2.1.2 will review those previous estimators.

2.1.1 Maximum *a posteriori* estimation

The simplest and most common estimator is the maximum a posteriori (MAP) estimator, which returns the parameters $\hat{g}$ that are most probable *a posteriori*:

$$\begin{aligned}\hat{g}_{\mathrm{MAP}}(\boldsymbol{x}) &\stackrel{\text{def}}{=} \operatorname*{argmax}_{g} p(\boldsymbol{g} \mid \boldsymbol{x}) \\ &= \operatorname*{argmax}_{g} p(\boldsymbol{x}, \boldsymbol{g}) \\ &= \operatorname*{argmax}_{g} \sum_{y \in \mathcal{Y}} p(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{g}) \\ &= \operatorname*{argmax}_{g} p(\boldsymbol{g}) \sum_{y \in \mathcal{Y}} p(\boldsymbol{y} \mid \boldsymbol{g}) p(\boldsymbol{x} \mid \boldsymbol{y}). \end{aligned} \tag{2.2}$$

In the special case where $p(\boldsymbol{g})$ is constant in $\boldsymbol{g}$ (a uniform prior), this reduces to the maximum likelihood estimator.

In the case of grammar induction, $\boldsymbol{x}$ is an observed corpus, and $\mathcal{Y}$ is the space of all possible treebanks. For each $\boldsymbol{g}$, Equation (2.2) requires a summation over combinatorially many treebanks $\boldsymbol{y}$ that are consistent with the observed corpus. The good news is that if $\boldsymbol{g}$ is a PCFG (Section 1.3), then this summation can be performed in time that is polynomial in the length of the corpus, using the inside algorithm (Baker, 1979). The bad news is that the sum is not convex in the parameters $\boldsymbol{g}$, and thus it is difficult to maximize Equation (2.2). In fact, even if we simplify $\sum$ to max (the Viterbi approximation) and use the uniform prior, the max over $\boldsymbol{g}$ is NP-hard to compute and inapproximable (Cohen and Smith, 2010).

In practice, it is common to compute a *local* maximum, using either gradient ascent or the expectation-maximization (EM) algorithm (Dempster, Laird, and Rubin, 1977;

Baker, 1979; Lari and Young, 1990). Recently, thanks to the development of auto-differentiation libraries (Paszke et al., 2017; Abadi et al., 2015), it is straightforward to enlist back-propagation to simplify both of these methods (Eisner, 2016), as long as $p(\boldsymbol{g} \mid \boldsymbol{X})$ has an analytical form. Because maximization is only local, careful initialization can be an effective way to inject prior knowledge, even when the actual prior distribution is uniform (Klein and Manning, 2004).

For more sophisticated priors like Dirichlet (Smith, 2006; Naseem et al., 2010; Kurihara and Sato, 2004) or logistic normal (Cohen, Gimpel, and Smith, 2009), the summation needed to compute $p(\boldsymbol{g} \mid \boldsymbol{X})$ may no longer be efficiently computable. In this case, one option is to instead locally maximize a variational lower bound on Equation (2.2) (Kurihara and Sato, 2004; Cohen, Gimpel, and Smith, 2009).

2.1.2 Bayes estimator

A generalization is to use a *Bayes estimator*. Let $\text{Loss}(\boldsymbol{g}, \boldsymbol{g}^*)$ be a given function that should be used to evaluate the error in an estimate $\boldsymbol{g}$ relative to the true parameters $\boldsymbol{g}^*$. The *Bayes risk* R is the function defined by $\text{R}(\hat{\boldsymbol{g}}) \overset{\text{def}}{=} \mathop{\mathbb{E}}_{\boldsymbol{X},\boldsymbol{G}}[\text{Loss}(\hat{\boldsymbol{g}}(\boldsymbol{X}), \boldsymbol{G})]$ for each estimator $\hat{\boldsymbol{g}}$, where the expectation is taken under our Bayesian model Equation (2.1). The Bayes estimator $\hat{\boldsymbol{g}}_{\text{BAY}}$ is defined by

$$\begin{aligned} \hat{\boldsymbol{g}}_{\text{BAY}} &\overset{\text{def}}{=} \operatorname*{argmin}_{\hat{\boldsymbol{g}}} \text{R}(\hat{\boldsymbol{g}}) \\ &= \operatorname*{argmin}_{\hat{\boldsymbol{g}}} \mathop{\mathbb{E}}_{\boldsymbol{X},\boldsymbol{G}}[\text{Loss}(\hat{\boldsymbol{g}}(\boldsymbol{X}), \boldsymbol{G})]. \end{aligned} \tag{2.3}$$

In other words, $\hat{\boldsymbol{g}}_{\text{BAY}}$ minimizes the Bayes risk among all estimators, which is challenging to find because a brute-force search over all possible functions is obviously intractable.

An equivalent definition (Lehmann and Casella, 2006) is to say that a Bayes estimator is one that, given $\boldsymbol{x}$, finds a minimum-risk estimate of $\boldsymbol{g}$:

$$\hat{\boldsymbol{g}}_{\mathrm{BAY}}(\boldsymbol{x}) = \operatorname*{argmin}_{\boldsymbol{g}} \mathop{\mathbb{E}}_{\boldsymbol{G}|\boldsymbol{x}} [\mathrm{Loss}(\boldsymbol{g}, \boldsymbol{G})] \text{ (for any } \boldsymbol{x}) \tag{2.4}$$

A special case is the 0-1 loss function where $\mathrm{Loss}(\boldsymbol{g}, \boldsymbol{g}^*)$ is 0 if $\boldsymbol{g} = \boldsymbol{g}^*$ and 1 otherwise. In this case, (2.4) reduces to the MAP estimator. Below, we discuss other special cases.

2.1.3 Posterior mean estimator

As shown in Equation (2.4), in contrast to the MAP approach which is interested in *a* distribution with the optimal posterior probability, $\hat{\boldsymbol{g}}_{\mathrm{BAY}}(\boldsymbol{x})$ pays attention to the expected loss over the *entire* posterior distribution $p(\boldsymbol{G} \mid \boldsymbol{x})$. While the general form of $\hat{\boldsymbol{g}}_{\mathrm{BAY}}$ includes an argmax operator that is computationally intractable (for problems like grammar induction), a carefully designed loss function $\mathrm{Loss}(\cdot, \cdot)$ may circumvent this difficulty without optimization. The posterior mean (PM) estimator is such a design where $\mathrm{Loss}(\cdot, \cdot)$ is a squared error:

$$\hat{\boldsymbol{g}}_{\mathrm{PM}}(\boldsymbol{x}) = \operatorname*{argmin}_{\hat{\boldsymbol{g}}'} \mathop{\mathbb{E}}_{\boldsymbol{G}|\boldsymbol{x}} [(\hat{\boldsymbol{g}}' - \boldsymbol{G})^2] \tag{2.5}$$

$$= \mathop{\mathbb{E}}_{\boldsymbol{G}|\boldsymbol{x}} [\boldsymbol{G}]. \tag{2.6}$$

Equation (2.6) is obtained by setting the derivative of $\hat{\boldsymbol{g}}'$ in Equation (2.5) to be $\mathbf{0}$. Compared to MAP, PM doesn't require optimization, thereby avoiding the issue of local optima (Cohen and Smith, 2012) during the local search. The standard technique for approximating the expectation is the Monte Carlo integral, which averages over sampled instances from the posterior distribution $p(\boldsymbol{G} \mid \boldsymbol{x})$. Cohen, Gimpel, and

Smith (2009) and Kurihara and Sato (2004) sample from a variational Bayes (VB) approximation to the posterior, while Johnson, Griffiths, and Goldwater (2007) sample approximately using a Markov chain Monte Carlo (MCMC) method.

2.2 Amortized Bayes Estimator (our proposal)

We propose to return to the formulation in Equation (2.3), but to limit the search over estimators to a parametric family $\mathcal{E} = \{\mathrm{e}_{\mathbf{\Theta}}\}_{\mathbf{\Theta} \in \mathbb{R}^d}$, which is parameterized by a d-dimensional real-valued vector $\mathbf{\Theta}$. We can now (locally) minimize the Bayes risk within this family by optimizing the parameter vector $\mathbf{\Theta}$. Using the methods of deep learning, we can construct a highly expressive family $\mathcal{E}$ and find high-quality local minima—so we have reason to hope that our estimator will perform nearly as well as a Bayes estimator. This parameterization has also been used in other machine learning methods like the Wasserstein GAN (Arjovsky, Chintala, and Bottou, 2017).

This gives us an estimator

$$\hat{g} = \operatorname*{argmin}_{\hat{g} \in \mathcal{E}} \mathop{\mathbb{E}}_{\boldsymbol{X},\boldsymbol{G}} [\mathrm{Loss}(\hat{g}(\boldsymbol{X}), \boldsymbol{G})] \tag{2.7}$$

$$= \mathrm{e}_{\mathbf{\Theta}^*}, \tag{2.8}$$

where

$$\mathbf{\Theta}^* = \operatorname*{argmin}_{\mathbf{\Theta} \in \mathbb{R}^d} \mathop{\mathbb{E}}_{\boldsymbol{X},\boldsymbol{G}} [\mathrm{Loss}(\mathrm{e}_{\mathbf{\Theta}}(\boldsymbol{X}), \boldsymbol{G})] \tag{2.9}$$

$$\approx \operatorname*{argmin}_{\mathbf{\Theta} \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^{N} \mathrm{Loss}(\mathrm{e}_{\mathbf{\Theta}}(\boldsymbol{x}_i), \boldsymbol{g}_i). \tag{2.10}$$

Our proposed *amortized Bayes (*AB*) estimator* $\hat{g}_{\text{AB}}$ arises from replacing Equation (2.9) with Equation (2.10), its empirical estimate based on a sample of N $(\boldsymbol{x}, \boldsymbol{g})$ pairs drawn from the marginal joint distribution $p(\boldsymbol{X}, \boldsymbol{G})$. This formulation converts our task of estimator construction to a "supervised" learning problem where $\boldsymbol{\Theta}^*$ is learned from N training examples.[2]

Algorithm 1 gives a general procedure for $\hat{g}_{\text{AB}}$, with the comments phrased in terms of the grammar induction setting. The procedure has 4 steps: (1) generate a set $\mathcal{L}$ which contains many (corpus, distribution) pairs from sampled the prior, (2) design a parametric prediction architecture $\mathrm{e}_{\boldsymbol{\Theta}}$ such as a neural network, (3) find good parameters $\boldsymbol{\Theta}^*$ by training on $\mathcal{L}$, and (4) for any given corpus $\boldsymbol{x}$, predict $\mathrm{e}_{\boldsymbol{\Theta}^*}(\boldsymbol{x})$ as the AB estimate. Note that once $\boldsymbol{\Theta}^*$ is trained, $\mathrm{e}_{\boldsymbol{\Theta}^*}(\cdot)$ is ready to evaluate any new $\boldsymbol{x}$ by mere forward computing at test time.

Algorithm 1 The estimation process using the Amortized Bayes estimator.

```
Input: observed corpus $x$; sample size $N$; loss function $\mathrm{Loss}(g, g^*)$
Output: $\hat{g}_{\text{AB}}(x)$, which is the distribution estimated by the AB estimator
1: for $i = 1$ to $N$ do
2:     $g_i \sim p(\cdot)$                              ▷ Sample a distribution $g_i$ from the prior
3:     $y_i, x_i \sim p(\cdot \mid g_i)$                ▷ Sample a treebank $y_i$ from $g_i$ and extract the fringe $x_i$
4: $\Theta^* \leftarrow \operatorname{argmin}_{\Theta \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^{N} \mathrm{Loss}(\mathrm{e}_{\Theta}(x_i), g_i)$      ▷ Find $\Theta^*$ by optimization
5: return $\mathrm{e}_{\Theta^*}(x)$
```

$\hat{g}_{\text{AB}}$ resembles an artificial linguist (Section 1.4) characterized by $\boldsymbol{\Theta}$, who gains "experience" by annotating many (N) training languages, eventually arriving at an approximation to $\boldsymbol{\Theta}^*$. This is different from the traditional estimators like $\hat{g}_{\text{PM}}$ and $\hat{g}_{\text{MAP}}$, which are stand-alone estimators (or per-example estimators) which evaluate every new language independently. It is analogous to a real linguist who gains

[2]It is possible to add a regularization term on $\boldsymbol{\Theta}$, which is suppressed from Equation (2.10).

"experience" from the domain knowledge in textbooks.[3] In our setting, the training languages are sampled from the prior $p(\boldsymbol{G})$ rather than from a textbook—but we will change that below (Section 2.4.5).

We call $\hat{\boldsymbol{g}}_{\mathrm{AB}}$ an *amortized Bayes* estimator in that $\boldsymbol{\Theta}^*$ is shared and auto-tuned to perform well across training corpora. In other words, when analyzing a test corpus $\boldsymbol{x}$, $\boldsymbol{\Theta}^*$ encodes experience from analyzing training corpora. Our approach is in the realm of *amortized inference* or *stochastic inverse learning* (Stuhlmüller, Taylor, and Goodman, 2013; Gershman and Goodman, 2014; Pakman and Paninski, 2018; Lee et al., 2019)—a technique of reusing parameters across datasets that has gained attention in the machine learning community. Their notable difference from ours is that most of them use $\boldsymbol{\Theta}$ to parameterize the posterior distribution for variational inference, while we directly parameterize the estimation method $\boldsymbol{g}$.

2.3 An Analogy: Statistical Estimation as Function Inversion

Bayesian statistics may be regarded as inverse probability. We are given a known probabilistic generative process that produced an observation $\boldsymbol{x}$ from an underlying parameter $\boldsymbol{g}$. We aim to invert that process to recover $\boldsymbol{g}$.

A simplified case arises in the familiar setting where we wish to invert a *deterministic function*. For simplicity and concreteness, let us consider a simple scalar function. Consider the equation $x = \sin(g)$, which deterministically computes an observable

[3]In linguistics, the contrast between ours and the previous estimators is loosely related to the difference between Greenberg's (Greenberg, 1963) and Chomsky's (Chomsky, 1965) approaches in research on *language universals*, where the former looks at a broad range of languages to gain generalization (like ours), while the later puts emphasis on the detailed, abstract study of a small number of languages (Comrie, 1989).

value x from an unknown value g. We would like to estimate the g that gave rise to x. (More precisely, we would like to find a possible g, since there may be multiple values g such that $x = \sin(g)$—just as in grammar induction.)

For this problem, there exists an efficient algorithm based on determining the Taylor series for the $\sin^{-1}$ (arcsine) function. This is analogous to statistical estimation problems that have a closed-form solution (e.g., the variance of a Gaussian distribution) or an algorithmically efficient solution (e.g., spectral estimation of hidden Markov models).

However, if no such algorithm existed, we could start with an initial guess for g and then optimize it by local search. Specifically, we could use the bisection method or Newton's method to seek a solution to $\sin(g) - x = 0$, or use gradient descent to seek a minimum of $(\sin(g) - x)^2$. This is analogous to the EM and gradient ascent methods reviewed in Sections 2.1.1 and 2.1.3.

Alternatively, a timeworn approach is to consult a "trigonometric table" of (x, g) pairs that records the result of precomputing $x = \sin(g)$ for many g values. We look up our observed x in the first column, finding a single pair (x_1, g_1) such that $x \approx x_1$, which implies that $x \approx \sin(g_1)$ as desired. For improved accuracy, we can identify two or more pairs $\{(x_i, g_i)\}$ such that $x \approx x_i$. We can then interpolate linearly or polynomially among these pairs, giving us an interpolated value $\hat{g}$ such that $x \approx \sin(\hat{g})$. This method can be regarded as constructing an approximate $\sin^{-1}$ function from the table of pairs (by piecewise polynomial regression, which is a simple example of a "local learning" algorithm in machine learning). Our AB estimator is constructed analogously: we similarly precompute a table of $(\boldsymbol{x}, \boldsymbol{g})$ pairs and then we use machine learning to fit a function $\hat{\boldsymbol{g}}_{\text{AB}}$ to them.

2.4 From Grammar Induction to Other Tasks

2.4.1 Grammar induction as Bayesian estimation

Grammar induction can be treated as a statistical estimation problem that can be addressed by our novel estimator $\hat{g}_{\text{AB}}$. We must assume that we have been given a general-purpose linguistic grammar: a collection of rules (or other elementary structures) that can be used to generate any sentence in any language. In a given language, some of these rules will have high weight, while some rules will be unused and have weight ≈ 0. We aim to estimate the vector $\boldsymbol{g}$ of weights for a given language, given a corpus $\boldsymbol{x}$ of that language and a prior distribution $p(\boldsymbol{G})$.

For decades, although linguists have proposed many grammar formalisms, the NLP grammar induction community has focused on estimating weights for PCFGs (Table 2.1). A PCFG is one type of grammar formalism. It uses a collection of context-free rules to define a family of language models, parameterized by the rule probabilities $\boldsymbol{g}$. As Section 2.1.1 noted, one particular advantage of the PCFG formalism in practice is that calculating the likelihood of PCFG parameters given a corpus can be done in polynomial time via dynamic programming (the inside algorithm).

2.4.2 Limitations of grammar induction

As discussed in Section 1.3, PCFGs have some difficulty capturing complex linguistic phenomena, in part because of their conditional independence assumptions.[4] Spitkovsky, Alshawi, and Jurafsky (2013) showed that even when a dependency-and-boundary (DBM) grammar (an extension of the DMV grammar) was estimated from

[4]Another issue is that PCFGs are designed to describe projective structures, whereas intuitive theories of German, Dutch and Czech make use of non-projective structures (Figure 1.3).

CFG rules	g
S $\rightarrow$ NP VP	0.9
S $\rightarrow$ VP NP	0.1
VP$\rightarrow$ VP PP	0.6
VP$\rightarrow$ V NP	0.4
NP$\rightarrow$ DET N	0.4
PP$\rightarrow$ P NP	0.000015
P $\rightarrow$ with	0.05

Table 2.1: A fragment of a PCFG that defines traditional constituency structures. If we preferred to use projective dependency structures, we could use a PCFG encoding of the popular dependency model with valence (DMV) (Klein and Manning, 2004), as explained by Klein and Manning (2004, Figure 4), Smith (2006, Figure 2.3) and Pate and Johnson (2016, Figure 1).

supervised data, it reached a mere 76.3 unlabelled attachment score (UAS)—far behind non-grammar-based parsers.[5] This illuminates the potential failings of a grammar estimation approach (at least for a relatively small grammar).

Since a more important goal for downstream tasks is to infer good parse structures $\boldsymbol{y}$, we do not wish to formulate our task as one of estimating grammar parameters $\boldsymbol{g}$. Estimating a grammar is unnecessary if it is only an indirect way of obtaining these parses—and if the grammar formalism is not powerful enough (as above), then even the best possible estimate of $\boldsymbol{g}$ may not lead to good parses.

Thus, our final parser (Chapter 5) is a discriminative, non-context-free parser that predicts parses directly without making use of an explicit grammar. The actual parsing architecture will be introduced in Chapter 5.

[5]The strong supervised parsers discussed in Section 1.3 are non-grammar-based parsers. Instead, they rely on feature-rich or neural models.

2.4.3 Unsupervised parsing as (amortized) Bayesian inference

We now explain how to do amortized Bayesian inference of parses, instead of amortized Bayesian estimation of the grammar.

The idea is simple. We sample training languages from the prior, just as in Algorithm 1. However, where we used to train an amortized system to predict each training language's *grammar* $\boldsymbol{g}$ from its corpus $\boldsymbol{x}$, we now train it to predict each training language's *treebank* $\boldsymbol{y}$, as shown in Figure 2.2. The training loss function $\mathrm{Loss}(\boldsymbol{g}, \boldsymbol{g}^*)$ is now replaced by a loss function on treebanks, $\mathrm{Loss}(\boldsymbol{y}, \boldsymbol{y}^*)$.

Algorithm 2 Amortized Bayes parsing of a corpus.

```
Input: observed corpus $x$; sample size $N$; loss function $\mathrm{Loss}(y, y^*)$
Output: $\hat{y}_{\mathrm{AB}}(x)$, which is the treebank estimated by the AB estimator
1: for $i = 1$ to $N$ do
2:     $g_i \sim p(\cdot)$                                        ▷ Sample a distribution $g_i$ from the prior
3:     $y_i, x_i \sim p(\cdot \mid g_i)$                ▷ Sample a treebank $y_i$ from $g_i$ and extract the fringe $x_i$
4: $\Theta^* \leftarrow \operatorname{argmin}_{\Theta \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^{N} \mathrm{Loss}(e_\Theta(x_i), y_i)$     ▷ Find $\Theta^*$ by optimization
5: return $e_{\Theta^*}(x)$
```

This reduces our unsupervised parsing problem to a supervised training problem on languages sampled from the prior.

Note that our trained parameter vector $\boldsymbol{\Theta}^*$ specifies a cross-linguistic or "universal" parser. It is trained to be able to parse *any* corpus $\boldsymbol{x}$ (from any language) into a treebank. It does this by examining $\boldsymbol{x}$ for surface cues to the deep structure of the language.

For this to work, $\boldsymbol{x}$ must be large enough to contain all the necessary surface cues. Our universal parser will not work properly on a single sentence $\boldsymbol{x}$, or even on a small corpus $\boldsymbol{x}$. If we dislike this state of affairs, we can fix it with a simple extension of our

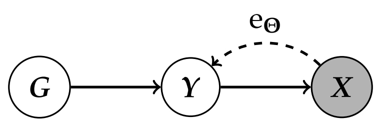


Figure 2.2: The setup of the proposed unsupervised parsing framework. Both corpus $\boldsymbol{X}$ is sampled from $\boldsymbol{G}$, and $\boldsymbol{G}$ is sampled from the prior $p(\boldsymbol{G})$. Instead of discovering $\boldsymbol{G}$, we want to discover the *latent treebank* $\boldsymbol{Y}$ from the observation $\boldsymbol{X}$ through a parameterized map $\mathrm{e}_{\boldsymbol{\Theta}}$ in the dashed arrow.

setup. The input $\boldsymbol{x}$ can be as small as desired—even a single sentence—provided that the parser is also given an additional unparsed corpus $\boldsymbol{u}$ of the same language, from which to extract the surface cues.

In other words, we change our generative model from Equation (2.1) to also generate "side information" $\boldsymbol{u}$:

$$p(\boldsymbol{G} = \boldsymbol{g}) \cdot p(\boldsymbol{Y} = \boldsymbol{y} \mid \boldsymbol{G} = \boldsymbol{g}) \cdot p(\boldsymbol{X} = \boldsymbol{x} \mid \boldsymbol{G} = \boldsymbol{g}, \boldsymbol{Y} = \boldsymbol{y}) \cdot \underbrace{p(\boldsymbol{U} = \boldsymbol{u} \mid \boldsymbol{G} = \boldsymbol{g})}_{\text{new step}} \tag{2.11}$$

The generative story for $\boldsymbol{u}$ in the final step may involve latent parses $\boldsymbol{y}'$ (see Footnote 1 on page 20), but this does not matter. In fact it is not even necessary for $\boldsymbol{u}$ to be a text corpus—formally speaking, it could be any side information about the language that is useful for extracting surface cues that are useful to the parser. (For example, a collection of n-gram counts, or a Swadesh list.) In Chapter 5, however, we will take $\boldsymbol{u}$ to be an unparsed text corpus of the language.

This gives rise to the following modified algorithm:

Algorithm 3 Amortized Bayes parsing of a corpus, with side information.

Input: observed corpus $\boldsymbol{x}$; sample size N; loss function $\mathrm{Loss}(\boldsymbol{y}, \boldsymbol{y}^*)$; side information $\boldsymbol{u}$

Output: $\hat{\boldsymbol{y}}_{\mathrm{AB}}(\boldsymbol{x})$, which is the treebank estimated by the AB estimator

```
1: for $i = 1$ to $N$ do
2:     $\boldsymbol{g}_i \sim p(\cdot)$                              ▷ Sample a distribution $\boldsymbol{g}_i$ from the prior
3:     $\boldsymbol{y}_i, \boldsymbol{x}_i \sim p(\cdot \mid \boldsymbol{g}_i)$     ▷ Sample a treebank $\boldsymbol{y}_i$ from $\boldsymbol{g}_i$ and extract the fringe $\boldsymbol{x}_i$
4:     $\boldsymbol{u}_i \sim p(\cdot \mid \boldsymbol{g}_i)$              ▷ Sample the side information for the language
5: $\boldsymbol{\Theta}^* \leftarrow \operatorname{argmin}_{\boldsymbol{\Theta} \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^{N} \mathrm{Loss}(\mathrm{e}_{\boldsymbol{\Theta}}(\boldsymbol{x}_i, \boldsymbol{u}_i), \boldsymbol{y}_i)$     ▷ Find $\boldsymbol{\Theta}^*$ by optimization
6: return $\mathrm{e}_{\boldsymbol{\Theta}^*}(\boldsymbol{x})$
```

2.4.4 Typology prediction as (amortized) Bayesian inference

The unsupervised parser in the previous section presumably extracts some kind of typological information from the given corpus to aid in parsing. For example, it must determine whether the language uses a basic word order that is SOV (subject-object-verb) or OSV (object-subject-verb) or something else. Typological information has previously been shown to be useful for both generative parsers (Naseem, Barzilay, and Globerson, 2012) and discriminative parsers (Täckström, McDonald, and Nivre, 2013; Zhang and Barzilay, 2015).

In Chapter 4, we will build a system for explicitly extracting fine-grained typological information from a text corpus. Again, this is not the grammar induction task since our system does not produce a full formal grammar, merely typological features.

Our typological information is syntactic: we assume that it can be extracted from a treebank $\boldsymbol{y}$, so we write it as $\boldsymbol{\tau}(\boldsymbol{y})$. Formally speaking, our problem is now just like unsupervised parsing (Algorithm 2), but it faces an easier problem because it does not have to predict $\boldsymbol{y}$ but only the summary information $\boldsymbol{\tau}(\boldsymbol{y})$. We now use a loss function $\mathrm{Loss}(\boldsymbol{\tau}, \boldsymbol{\tau}^*)$ on typological descriptions.

Algorithm 4 Amortized Bayes parsing of a corpus.

```
Input: observed corpus $\boldsymbol{x}$; sample size $N$; loss function $\text{Loss}(\boldsymbol{\tau}, \boldsymbol{\tau}^*)$
Output: $\hat{\boldsymbol{\tau}}_{\text{AB}}(\boldsymbol{x})$, which is the typological description estimated by the AB estimator
1: for $i = 1$ to $N$ do
2:     $\boldsymbol{g}_i \sim p(\cdot)$    ▷ Sample a distribution $\boldsymbol{g}_i$ from the prior
3:     $\boldsymbol{y}_i, \boldsymbol{x}_i \sim p(\cdot \mid \boldsymbol{g}_i)$    ▷ Sample a treebank $\boldsymbol{y}_i$ from $\boldsymbol{g}_i$ and extract the fringe $\boldsymbol{x}_i$
4: $\boldsymbol{\Theta}^* \leftarrow \underset{\boldsymbol{\Theta} \in \mathbb{R}^d}{\text{argmin}} \frac{1}{N} \sum_{i=1}^{N} \text{Loss}(\text{e}_{\boldsymbol{\Theta}}(\boldsymbol{x}_i), \boldsymbol{\tau}(\boldsymbol{y}_i))$    ▷ Find $\boldsymbol{\Theta}^*$ by optimization
5: return $\text{e}_{\boldsymbol{\Theta}^*}(\boldsymbol{x})$
```

2.4.5 Eliminating explicit grammars altogether

Notice that for the two tasks in Sections 2.4.3 and 2.4.4, the supervised training algorithms made no use of the sampled grammars $\boldsymbol{g}_i$. The grammars are discarded after being used to generate treebanks.

As a result, we do not actually need to sample grammars $\boldsymbol{g}_i$ from a prior $p(\boldsymbol{G})$. All that supervised training actually needs is the treebanks $\boldsymbol{y}_i$ (and any side information $\boldsymbol{u}_i$). If we have some non-grammar-based way to sample treebanks, we can use that method instead.

This move saves us from having to commit to any grammar formalism. The distribution from which we sample treebanks may be regarded as having implicit knowledge of some distribution over grammars, but the grammars are never formalized or made explicit.

In short, a *distribution over treebanks* is an alternative way to encode prior knowledge of the range of possible human languages that our typology predictors and universal parsers may encounter. For this reason, Chapters 3 and 6 will explore methods for sampling synthetic treebanks from possible human languages without use of a grammar. The synthetic treebanks from Chapter 3 will be used to train our typology predictor (Chapter 4) and our universal parser (Chapter 5).

2.5 Discussion

To sum up, the AB estimator has three advantages.

First, the family of parametric functions $\mathrm{e}_{\boldsymbol{\Theta}}$ for performing estimation or inference can be designed so that it is likely to contain a good function. We will adopt an expressive family.

Second, that family can be designed to be easily trained (in conjunction with the given task loss function $\mathrm{Loss}(\cdot,\cdot)$). For example, if we were to use log-linear models and a log-loss function, then finding $\boldsymbol{\Theta}^*$ would be a convex optimization problem with a single global maximum. In practice, we will use neural networks, which tend to find good local optima because they have many parameters that they can adjust to escape poor parameter settings.

Finally, $\mathrm{e}_{\boldsymbol{\Theta}^*}$ is a discriminative system. If the task is parsing or typology prediction, it can bypass the problem of grammar induction—we simply train it end-to-end to minimize expected task loss.

We close by reminding the reader why the AB estimator is necessary. Inasmuch as $\mathrm{e}_{\boldsymbol{\Theta}^*}$ is supposed to generalize well to any real languages at test time, training examples must reflect the true population of human languages. For our tasks in Chapters 4 and 5, the most straightforward approach would be to train on existing annotated languages. The most recent UD treebank (Nivre et al., 2019) which has 146 treebanks covering 83 languages. However, this only gives us 146 training examples—not enough for supervised training of an AB estimator that could generalize well. That is why the AB technique trains on a large sample of synthetic languages, which reflect prior knowledge about the distribution of possible human languages.

Following the approach in Section 2.4.5, we now turn to the problem of constructing good synthetic training examples $(\boldsymbol{x}_i, \boldsymbol{y}_i)$. In Chapter 3, we will introduce a procedure for generating synthetic treebanks given a collection of real treebanks such as the Universal Dependencies dataset. Our resulting set of synthetic treebanks, the *Galactic Dependencies* dataset, will serve as the main working data in this thesis.

Chapter 3

Resolving the Challenge of Data Sparsity—the Galactic Dependencies

This chapter focuses on the data sparsity problem referred to in Section 2.5. We introduce the Galactic Dependencies 1.0—a large set of synthetic languages not found on Earth, but annotated in Universal Dependencies format—to fill in the gaps among real languages. More generally, this new resource aims to provide training and development data for NLP methods that aim to adapt to unfamiliar languages. Each synthetic treebank is produced from a real treebank by stochastically permuting the dependents of nouns and/or verbs to match the word order of other real languages. We discuss the usefulness, realism, parsability, perplexity, and diversity of the synthetic languages. As a simple demonstration of the use of the Galactic Dependencies, we consider the *single-source selection parsing*, which attempts to parse a real target language using a parser trained on a "nearby" source language. We find that including synthetic source languages somewhat increases the diversity of the source pool, which significantly improves results for most target languages.

3.1 Motivation

As discussed in Section 2.5, we have formulated our framework on unsupervised parsing as a supervised learning problem that has very sparse data by machine learning standards—each of the IID training examples is an entire language. More broadly, this setup is applicable to an extensive range of NLP tasks:

- typological classification of a language on various dimensions;
- adaptation of any existing NLP system to new, low-resource languages;
- induction of a syntactic grammar from text;
- discovery of a morphological lexicon from text;
- other types of unsupervised discovery of linguistic structure.

Given a corpus or other data about a language, we might aim to predict whether it is an SVO language, or to learn to pick out its noun phrases. For such problems, a single training or test example corresponds to an entire human language.

Unfortunately, we usually have only from 1 to 100 languages (Section 2.5) to work with. In contrast, machine learning methods thrive on data, and recent AI successes have mainly been on tasks where one can train richly parameterized predictors on a huge set of IID (input, output) examples. Even 7,000 training examples—one for each language or dialect on Earth—would be a small dataset by contemporary standards.

As a result, it is challenging to develop systems that will discover structure in new languages in the same way that an image segmentation method, for example, will discover structure in new images. The limited resources even make it challenging to develop methods that handle new languages by unsupervised, semi-supervised, or transfer learning. Some such projects evaluate their methods on new sentences of the same languages that were used to develop the methods in the first place—which leaves one worried that the methods may be inadvertently tuned to the development languages and may not be able to discover correct structure in other languages. Other projects take care to hold out languages for evaluation (Spitkovsky, 2013; Cotterell, Peng, and Eisner, 2015), but then are left with only a few development languages on which to experiment with different unsupervised methods and their hyperparameters.

If we had many languages, then we could develop better unsupervised language learners. Even better, like our approach for unsupervised parsing formulated in Algorithm 2, we could treat the general linguistic structure discovery as a supervised learning problem. That is, we could train a system to extract features from the surface of a language that are predictive of its deeper structure. Principles & Parameters theory (Chomsky, 1981) conjectures that such features exist and that the juvenile human brain is adapted to extract them.

Our goal in this chapter is to release a set of about 50,000 high-resource languages that could be used to train supervised learners, or to evaluate less-supervised learners during development. These "unearthly" languages are intended to be at least *similar* to possible human languages. As such, they provide useful additional training and development data that is slightly out of domain (reducing the variance of a system's learned parameters at the cost of introducing some bias). The release is available at `https://github.com/gdtreebank/gdtreebank`.

In addition to releasing thousands of treebanks, we provide scripts that can be used to "translate" other annotated resources into these synthetic languages. E.g., given a corpus of English sentences labeled with sentiment, a researcher could reorder the words in each English sentence according to one of our English-based synthetic languages, thereby obtaining *labeled* sentences in the synthetic language.

3.2 Related Work

Synthetic data generation is a well-known method for effectively training a large model on a small dataset. Abu-Mostafa (1995) reviews early work that provided "hints" to a learning system in the form of virtual training examples. While datasets have grown in recent years, so have models: e.g., neural networks have many parameters to train. Thus, it is still common to create synthetic training examples—often by adding noise to real inputs or otherwise transforming them in ways that are expected to preserve their labels. Domains where it is easy to exploit these invariances include image recognition (Simard, Steinkraus, and Platt, 2003; Krizhevsky, Sutskever, and Hinton, 2012), speech recognition (Jaitly and Hinton, 2013; Cui, Goel, and Kingsbury, 2015), information retrieval (Vilares, Vilares, and Otero, 2011), grammatical error correction (Rozovskaya and Roth, 2010), and reduction of stereotypes (Zmigrod et al., 2019).

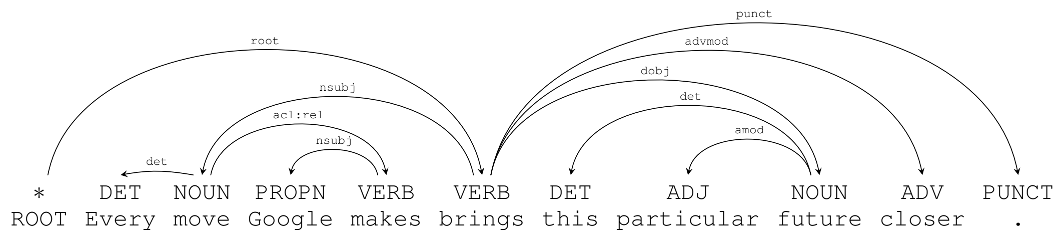
Synthetic datasets have also arisen recently for semantic tasks in natural language processing. bAbI is a dataset of facts, questions, and answers, generated by random simulation, for training machines to do simple logic (Weston et al., 2016). Hermann et al. (2015) generate reading comprehension questions and their answers, based on a large set of news-summarization pairs, for training machine readers. Wang, Berant, and Liang (2015a) generate synthetic (logical-form, sentence) pairs as training data for semantic parsers. Serban et al. (2016) used RNNs to generate 30 million factoid questions about Freebase, with answers, for training question-answering systems. Wang, Berant, and Liang (2015b) obtain data to train semantic parsers in a new domain by first generating synthetic (utterance, logical form) pairs and then asking human annotators to paraphrase the synthetic utterances into more natural human language.

In speech recognition, morphology-based "vocabulary expansion" creates synthetic word forms (Rasooli et al., 2014; Varjokallio and Klakow, 2016).

In machine translation, researchers have often tried to automatically preprocess parse trees of a source language to more closely resemble those of the target language, using either hand-crafted or automatically extracted rules (Dorr et al., 2002; Collins, Koehn, and Kucerova, 2005, etc.; see review by Howlett and Dras (2011)). More famously, the back-translation (Sennrich, Haddow, and Birch, 2016) trains foreign-to-local systems using synthetic parallel data generated from the trained local-to-foreign systems, which has become a well-received technique in neural machine translation (NMT).

3.3 Synthetic Language Generation

A treebank is a corpus of parsed sentences of some language. We propose to derive each synthetic treebank from some real treebank. By manipulating the existing parse trees, we obtain a useful corpus for our synthetic language—a corpus that is already *tagged*, *parsed*, and *partitioned* into training/development/test sets. Additional data in the synthetic language can be obtained, if desired, by automatically parsing additional real-language sentences and manipulating these trees in the same way.



Language	Sentence
English	Every move Google makes brings this particular future closer.
English[French/N]	Every move Google makes brings this future particular closer.
English[Hindi/V]	Every move Google makes this particular future closer brings.
English[French/N, Hindi/V]	Every move Google makes this future particular closer brings.

Figure 3.1: The original UD tree for a short English sentence, and its "translations" into three synthetic languages, which are obtained by manipulating the tree. (Moved constituents are underlined.) Each language has a different distribution over surface part-of-speech sequences.

3.3.1 Method

We begin with the Universal Dependencies collection version 1.2 (Nivre et al., 2015; Nivre et al., 2016),[1] or UD. This provides 37 manually edge-labeled dependency treebanks in 33 real languages, in a consistent style and format—the Universal Dependencies format. An example appears in Figure 3.1.

In this thesis, we select a *substrate* language S represented in the UD treebanks, and systematically reorder the dependents of some nodes in the S trees, to obtain trees of a *synthetic* language S'.

Specifically, we choose a *superstrate* language $R_{\texttt{V}}$, and write $S' = S[R_{\texttt{V}}/\texttt{V}]$ to denote a (projective) synthetic language obtained from S by permuting the dependents of verbs (V) to match the ordering statistics of the $R_{\texttt{V}}$ treebanks. We can similarly permute the dependents of nouns (N).[2] This permutes about 93% of S's nodes (Table 3.2),

[1] http://universaldependencies.org

[2] In practice, this means applying a single permutation model to permute the dependents of every word tagged as NOUN (common noun), PROPN (proper noun), or PRON (pronoun).

as UD treats adpositions and conjunctions as childless dependents.

For example, English[French/N, Hindi/V] is a synthetic language based on an English substrate, but which adopts subject-object-verb (SOV) word order from the Hindi superstrate and noun-adjective word order from the French superstrate (Figure 3.1). Note that it still uses English lexical items.

Our terms "substrate" and "superstrate" are borrowed from the terminology of creoles, although our synthetic languages are unlike naturally occurring creoles. Our substitution notation $S' = S[R_{\mathtt{N}}/\mathtt{N}, R_{\mathtt{V}}/\mathtt{V}]$ is borrowed from the logic and programming languages communities.

3.3.2 Discussion

There may be more adventurous ways to manufacture synthetic languages (see Section 3.8 for some options). However, we emphasize that our current method is designed to produce fairly *realistic* languages.

First, we retain the immediate dominance structure and lexical items of the substrate trees, altering only their linear precedence relations. Thus each sentence remains topically coherent; nouns continue to be distinguished by case according to their role in the clause structure; *wh*-words continue to c-command gaps; different verbs (e.g., transitive vs. intransitive) continue to be associated with different subcategorization frames; and so on. For example, in Figure 3.1, "particular" in "English[French/N]" continues to be the adjectival modifier of "future", and "closer" in "English[French/N, Hindi/V]" continues to be the adverbial modifier of "brings". These important properties would not be captured by a simple context-free model of dependency trees, which is why we modify real sentences rather than generating new sentences from such a model. In addition, our method obviously preserves the basic context-free properties, such as the fact that verbs typically subcategorize for one or two nominal arguments (Naseem et al., 2010).

Second, by drawing on real superstrate languages, we ensure that our synthetic languages use plausible word orders. For example, if R_{V} is a V2 language that favors SVO word order but also allows OVS, then S' will match these proportions. Similarly, S' will place adverbs in reasonable positions with respect to the verb.

We note, however, that our synthetic languages might violate some typological universals or typological tendencies. For example, R_{V} might prescribe head-initial verb orderings while R_{N} prescribes head-final noun orderings, yielding an unusual language. Worse, we could synthesize a language that uses free word order (from R_{V}) even though nouns (from S) are not marked for case. Such languages are rare, presumably for the functionalist reason that sentences would be too ambiguous. One could automatically filter out such an implausible language S', or downweight it, upon discovering that a parser for S' was much less accurate on held-out data than a comparable parser for S.

We also note that our reordering method (Section 3.4) does ignore some linguistic structure. For example, we do not currently condition the order of the dependent subtrees on their heaviness or on the length of resulting dependencies, and thus we will not faithfully model phenomena like heavy-shift (Hawkins, 1994; Eisner and Smith, 2010). Nor will we model the relative order of adjectives. We also treat all verbs interchangeably, and thus use the same word orders—drawn from R_{V}—for both main clauses and embedded clauses. This means that we will never produce a language like German (which uses V2 order in main clauses and SOV order in embedded clauses), even if $R_{\mathrm{V}} =$ German. All of these problems could be addressed by enriching the features (for example, using syntactically refined the tag set through parent annotation or supertagging) that are described in the next section.

3.4 Modeling Dependent Order

Let A be a part-of-speech tag, such as `VERB`. To produce a dependency tree in language $S' = S[R_A/A]$, we start with a projective dependency tree in language S.[3] For each node a in the tree that is tagged with A, we stochastically select a new ordering for its dependent nodes, including a position in this ordering for the head a itself. Thus, if node a has $n-1$ dependents, then we must sample from a probability distribution over $n!$ orderings.

Our job in this section is to define this probability distribution. Using $o = (o_1, \ldots, o_n)$ to denote an ordering of these n nodes, we define a log-linear model over the possible values of o:

$$p_{\boldsymbol{\theta}}(o \mid a) = \frac{1}{Z(a)} \exp \sum_{1 \le i < j \le n} \boldsymbol{\theta} \cdot \mathbf{f}(o, i, j) \tag{3.1}$$

Here $Z(a)$ is the normalizing constant for node a. $\boldsymbol{\theta}$ is the parameter vector of the model. $\mathbf{f}$ extracts a sparse feature vector that describes the ordered pair of nodes o_i, o_j, where the ordering o would place o_i to the left of o_j.

[3] Our method can only produce projective trees. This is because it recursively generates a node's dependent subtrees, one at a time, in some chosen order. Thus, to be safe, we only apply our method to trees that were originally projective. See Section 3.8.

3.4.1 Efficient sampling

To sample exactly from the distribution $p_{\boldsymbol{\theta}}$,[4] we must explicitly compute all $n!$ unnormalized probabilities and their sum $Z(a)$.

Fortunately, we can compute each unnormalized probability in just $O(1)$ amortized time, if we enumerate the $n!$ orderings o using the Steinhaus-Johnson-Trotter algorithm (Sedgewick, 1977).[5] This enumeration sequence has the property that any two consecutive permutations o, o' differ by only a single swap of some pair of adjacent nodes. Thus their probabilities are closely related: the sum in equation Equation (3.1) can be updated in $O(1)$ time by subtracting $\boldsymbol{\theta} \cdot \mathbf{f}(o, i, i+1)$ and adding $\boldsymbol{\theta} \cdot \mathbf{f}(o', i, i+1)$ for some i. The other $O(n^2)$ summands are unchanged.

In addition, if $n \geq 8$, we avoid this computation by omitting the entire tree from our treebank; so we have at most $7! = 5040$ summands.

3.4.2 Training parameters on a real language

Our feature functions (Section 3.4.4) are fixed over all languages. They refer to the 17 node labels (POS tags) and 40 edge labels (dependency relations) that are used consistently throughout the UD treebanks.

For each UD language ℓ and each POS tag A, we find parameters $\boldsymbol{\theta}_A^\ell$ that globally maximize the unregularized log-likelihood:

$$\boldsymbol{\theta}_A^\ell = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_a \log p_{\boldsymbol{\theta}}(o_a \mid a) \tag{3.2}$$

[4]We could alternatively have used MCMC sampling.

[5]A clean Python implementation by David Eppstein could be found at `www.ics.uci.edu/~eppstein/PADS/Permutations.py`. In addition, Brent Yorgey gives a nice explanation at `www.mathlesstraveled.com/2013/01/03/the-steinhaus-johnson-trotter-algorithm/`.

Here a ranges over all nodes tagged with A in the projective training trees of the ℓ treebank, omitting nodes with $n \geq 8$ for speed. o_a is the ordering of a's nodes, which is observed in the treebank.

The expensive part of this computation is the gradient of $\log Z(a)$, which is an expected feature vector. To compute this expectation efficiently, we again take care to loop over the permutations in Steinhaus-Johnson-Trotter order.

A given language ℓ may not use all of the tags and relations. Universal features that mention unused tags or relations do not affect Equation (3.2), and their weights remain at 0 during training.

3.4.3 Setting parameters of a synthetic language

We use Equation (3.1) to permute the A nodes of substrate language S into an order resembling superstrate language R_A. In essence, this applies the R_A ordering model to *out-of-domain data*, since the A nodes may have rather different sets of dependents in the S treebank than in the R_A treebank. We mitigate this issue in two ways.

First, our ordering model Equation (3.1) is designed to be more robust to transfer than, say, a Markov model. The position of each node is influenced by all $n-1$ other nodes, not just by the two adjacent nodes. As a result, the burden of explaining the ordering is distributed over more features, and we hope some of these features will transfer to S. For example, suppose R_A lacks adverbs and yet we wish to use $\boldsymbol{\theta}_A^{R_A}$ to permute a sequence of S that contains adverbs. Even though the resulting order must disrupt some familiar non-adverb bigrams by inserting adverbs, other features—which consider non-adjacent tags—will still favor an R_A-like order for the non-adverbs.

Second, we actually sample the reordering from a distribution $p_{\boldsymbol{\theta}}$ with an interpolated parameter vector

$$\boldsymbol{\theta} = \boldsymbol{\theta}_A^{S'} = (1-\lambda)\boldsymbol{\theta}_A^{R_A} + \lambda\boldsymbol{\theta}_A^{S}, \tag{3.3}$$

where $\lambda = 0.05$. This gives a weighted product of experts, in which ties are weakly broken in favor of the substrate ordering. (Ties arise when R_A is unfamiliar with some tags that appear in S, e.g., adverb.)

3.4.4 Feature templates

We write t_i for the POS tag of node o_i, and r_i for the dependency relation of o_i to the head node. If o_i is itself the head, then necessarily $t_i = A$,[6] and we specially define $r_i = \texttt{head}$.

In our feature vector $\mathbf{f}(o, i, j)$, the features with the following names have value 1, while all others have value 0:

- $\texttt{L.}t_i\texttt{.}r_i$ and $\texttt{L.}t_i$ and $\texttt{L.}r_i$, provided that $r_j = \texttt{head}$. For example, `L.ADJ` will fire on each `ADJ` node to the left of the head.

- $\texttt{L.}t_i\texttt{.}r_i\texttt{.}t_j\texttt{.}r_j$ and $\texttt{L.}t_i\texttt{.}t_j$ and $\texttt{L.}r_i\texttt{.}r_j$, provided that $r_i \neq \texttt{head}, r_j \neq \texttt{head}$. These features detect the relative order of two siblings.

- $d.t_i.r_i.t_j.r_j$, $d.t_i.t_j$, and $d.r_i.r_j$, where d is `l` (left), `m` (middle), or `r` (right) according to whether the head position h satisfies $i < j < h$, $i < h < j$, or $h < i < j$. For example, `l.nsubj.dobj` will fire on SOV clauses. This is a specialization of the previous feature, and is skipped if $i = h$ or $j = h$.

[6]Recall that for each head POS A of language ℓ, we learn a *separate* ordering model with parameter vector $\boldsymbol{\theta}_A^{\ell}$.

- A . t_i . r_i . t_j . r_j and A . t_i . t_j and A . r_i . r_j, provided that $j = i+1$. These "bigram features" detect two adjacent nodes. For this feature and the next one, we extend the summation in Equation (3.1) to allow $0 \le i < j \le n+1$, taking $t_0 = r_0 =$ BOS ("beginning of sequence") and $t_{n+1} = r_{n+1} =$ EOS ("end of sequence"). Thus, a bigram feature such as A.DET.EOS would fire on DET when it falls at the end of the sequence.

- H . t_i . r_i . t_{i+1} . r_{i+1} . . . . . t_j . r_j, provided that $i+2 \le j \le i+4$. Among features of this form, we keep only the 10% that fire most frequently in the training data. These "higher-order k-gram" features memorize sequences of lengths 3 to 5 that are common in the language.

Notice that for each non-H feature that mentions both tags t and relations r, we also defined two *backoff* features, omitting the t fields or r fields respectively.

To illuminate, Table 3.1 uses the examples in Figure 3.1 and compares the features of the two subtrees.

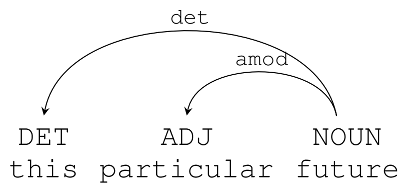
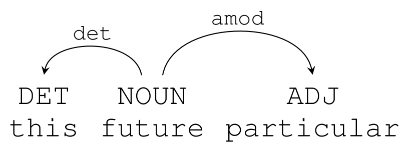
Template		
L.t_i.r_i	L.DET.det, L.ADJ.amod	L.DET.det
L.t_i.r_i.t_j.r_j	L.DET.det.ADJ.amod	-
d.t_i.r_i.t_j.r_j	l.DET.det.ADJ.amod	m.DET.det.ADJ.amod
A.t_1.r_1.t_2.r_2	A.BOS.BOS.DET.det, A.DET.det.ADJ.amod, A.ADJ.amod.NOUN.head, A.NOUN.head.EOS.EOS	A.BOS.BOS.DET.det, A.DET.det.NOUN.head, A.NOUN.head.ADJ.amod, A.ADJ.amod.EOS.EOS

Table 3.1: Features that fire in the two subtrees

plus backoff features and H features (not shown).

3.5 The Resource

In our current version (GD v1.0), we release real and synthetic treebanks based on UD v1.2. Each synthetic treebank is a modified work that is freely licensed under the same CC or GPL license as its substrate treebank. We provide *all* languages of the form S, $S[R_{\texttt{V}}/\texttt{N}]$, $S[R_{\texttt{N}}/\texttt{V}]$, and $S[R_{\texttt{N}}/\texttt{N}, R_{\texttt{V}}/\texttt{V}]$, where the substrate S and the superstrates $R_{\texttt{N}}$ and $R_{\texttt{V}}$ each range over the 37 available treebanks that represent 33 languages. ($R_{\texttt{N}} = S$ or $R_{\texttt{V}} = S$ gives "self-permutation"). This yields $37 \times 38 \times 38 = 53{,}428$ languages in total.

Each language is provided as a directory of 3 files: training, development, and test treebanks. The directories are systematically named: for example, English[French/N, Hindi/V] can be found in directory `en~fr@N~hi@V`. Our treebanks provide alignment information, to facilitate error analysis as well as work on machine translation. Each word in a synthetic sentence is annotated with its original position in the substrate sentence. Thus, all synthetic treebanks derived from the same substrate treebank are node-to-node aligned to the substrate treebank and hence to one another.

In addition to the generated data, we also provide the parameters θ_A^ℓ of our ordering models; code for training new ordering models; and code for producing new synthetic trees and synthetic languages. Our code should produce reproducible results across platforms, thanks to Java's portability and our standard random number seed of 0.

lang	sents	tokens	T	UAS	R
ar	4K / 6K	119K / 226K	85%	72% / 69%	0.37
cs	5K / 7K	687K / 1173K	94%	81% / 78%	0.38
de	9K / 14K	136K / 270K	94%	84% / 80%	0.47
es	10K / 14K	211K / 382K	94%	85% / 82%	0.32
fr	8K / 15K	154K / 356K	95%	86% / 84%	0.27
hi	9K / 13K	160K / 281K	96%	82% / 82%	0.20
it	9K / 12K	144K / 249K	95%	87% / 84%	0.30
la_itt	7K / 15K	87K / 247K	90%	66% / 58%	0.72
no	11K / 16K	135K / 245K	93%	82% / 79%	0.31
pt	5K / 9K	87K / 202K	96%	86% / 84%	0.32

Table 3.2: Some statistics on the 10 real training languages. When two numbers are separated by "/", the second represents the full UD treebank, and the first comes from our GD version, which discards non-projective trees and high-fanout trees ($n \geq 8$). UAS is the language's *parsability*: the unlabeled attachment score on its dev sentences after training on its train sentences. T is the percentage of GD tokens that are touched by reordering (namely `N`, `V`, and their dependents). $R \in [0, 1]$ measures the *freeness* of the language's word order, as the conditional cross-entropy of our trained ordering model p_{θ} relative to that of a uniform distribution: $R = \frac{H(\tilde{p}, p_{\theta})}{H(\tilde{p}, p_{\text{unif}})} = \frac{\text{mean}_x[-\log_2 p_{\theta}(o^*(a)|a)]}{\text{mean}_x[-\log_2 1/n(a)!]}$, where a ranges over all `N` and `V` tokens in the dev sentences, $n(a)$ is 1 + the number of dependents of a, and $o^*(a)$ is the observed ordering at a.

3.6 Exploratory Data Analysis

How do the synthetic languages compare to the real ones? For analysis and experimentation, we partition the real UD languages into train/dev/test.[7] Table 3.3 shows the split information of the 37 UD treebanks, attached by the their corresponding languages and the (sub-)family information according to http://universaldependencies.org. Following the usual setting of rich-to-poor transfer, we take the 10 largest non-English languages as our pool of real source languages, which we can combine to synthesize new languages. The remaining languages are the low-resource target languages, from which we randomly hold out 15 non-English languages as the test languages. During development, we studied and graphed performance on the remaining

[7] This is orthogonal to the train/dev/test split of each language's treebank.

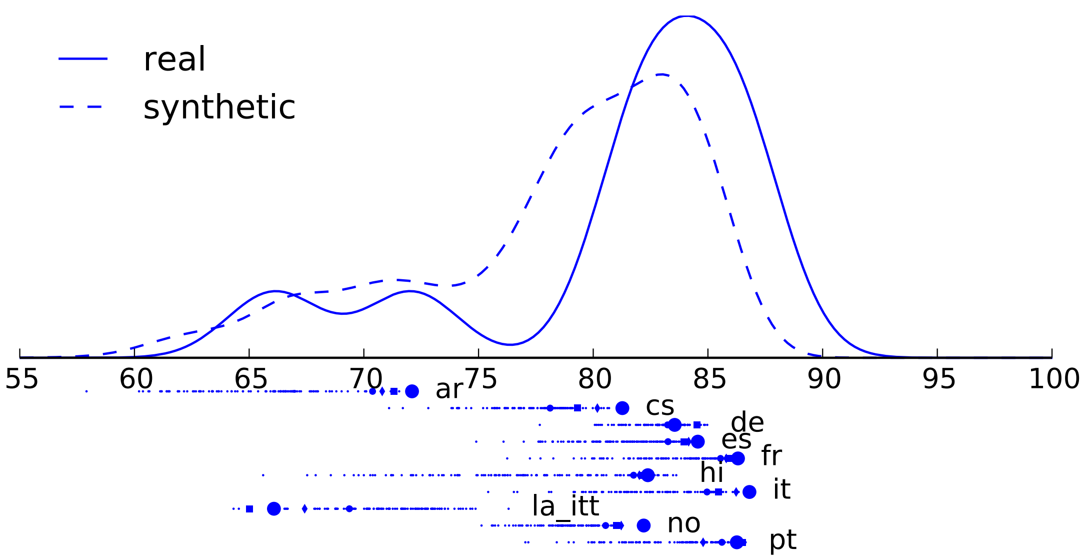


Figure 3.2: Parsability of real versus synthetic languages (defined as in Table 3.2). The upper graphs are kernel density estimates. Each lower graph is a 1-dimensional scatterplot, showing the parsability of some real language S (large dot) and all its permuted versions, including the "self-permuted" languages $S[S/\texttt{N}]$ (diamond), $S[S/\texttt{V}]$ (square), and $S[S/\texttt{N}, S/\texttt{V}]$ (medium dot).

8 languages—including English for interpretability. Table 3.2 shows some properties of the real training languages.

Throughout this chapter, we use the Yara parser (Rasooli and Tetreault, 2015), a fast arc-eager transition-based projective dependency parser, with beam size of 8. We train only *delexicalized* parsers, whose input is the sequence of POS tags. Following most previous works on unsupervised parsing, we evaluate parsing accuracy by the unlabeled attachment score (UAS), that is, the fraction of word tokens in held-out (dev) data that are assigned their correct parent. For language modeling, we train simple trigram backoff language models with add-1 smoothing, and we measure predictive accuracy as the perplexity of held-out (dev) data.

Figure 3.2–3.3 show how the parsability and perplexity of a real training language usually get worse when we permute it. We could have discarded low-parsability

Family	Sub-Family	Language (Treebank ID)	Split
Indo-European	Germanic	German (de)	Train
		Norwegian (no)	Train
		Danish (da)	Dev
		Dutch (nl)	Dev
		English (en)	Dev
		Gothic (got)	Dev
		Swedish (sv)	Test
	Slavic	Czech (cs)	Train
		Bulgarian (bg)	Dev
		Croatian (hr)	Test
		Old_Church_Slavonic (cu)	Test
		Polish (pl)	Test
		Slovenian (sl)	Test
	Romance	French (fr)	Train
		Italian (it)	Train
		Portuguese (pt)	Train
		Spanish (es)	Train
		Romanian (ro)	Test
	Greek	Ancient_Greek (grc)	Dev
		Ancient_Greek (grc_proiel)	Dev
		Greek (el)	Test
	Latin	Latin (la_itt)	Train
		Latin (la_proiel)	Dev
		Latin (la)	Test
	Celtic	Irish (ga)	Test
	Indic	Hindi (hi)	Train
	Iranian	Persian (fa)	Test
Uralic	Finnic	Estonian (et)	Dev
		Finnish (fi)	Dev
		Finnish (fi_ftb)	Test
	Ugric	Hungarian (hu)	Test
Afro-Asiatic	Semitic	Arabic (ar)	Train
		Hebrew (he)	Test
Austronesian	-	Indonesian (id)	Test
Basque	-	Basque (eu)	Test
Dravidian	Southern	Tamil (ta)	Test
Japanese	-	Japanese (ja_ktc)	Test

Table 3.3: Information of the 37 UD treebanks. As we are interested in transfer to *unseen* languages, in the following sections, any evaluation of Ancient_Greek as a language is computed by averaging the individual scores of "grc" and "grc_proiel" in blue, which will be referred as "grc" henceforth. Treebanks are marked in red are omitted for evaluation, because their languages are among the "Train" split as well. fi_ftb is also omitted because fi is in "Dev". Note that this setting is different from the original paper (Wang and Eisner, 2016), where the seen languages are also evaluated.

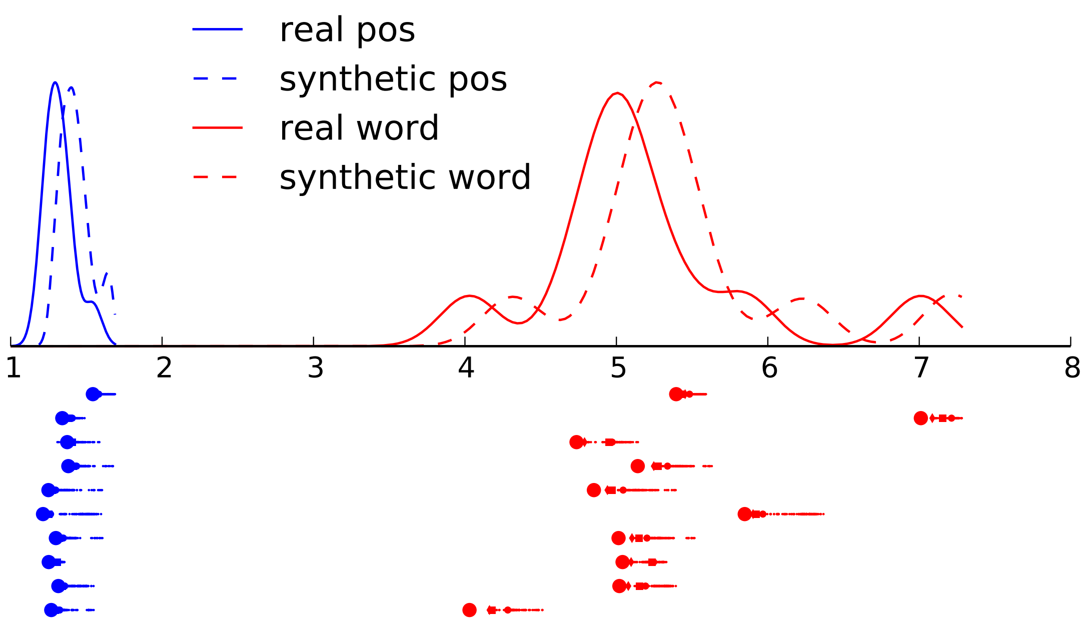


Figure 3.3: Perplexity of the POS sequence, as well as the word sequence, of real versus synthetic languages. Words with count < 10 are mapped to an OOV symbol.

synthetic languages, on the functionalist grounds that they would be unlikely to survive as natural languages anywhere in the galaxy. However, the curves in these figures show that most synthetic languages have parsability and perplexity within the plausible range of natural languages, so we elected to simply keep all of them in our collection.

An interesting exception in Figure 3.2 is Latin (la_itt), whose poor parsability—at least by a delexicalized parser that does not look at word endings—may be due to its especially free word order (Table 3.2). When we impose another language's more consistent word order on Latin, it becomes more parsable. Elsewhere, permutation generally hurts, perhaps because a real language's word order is globally optimized to enhance parsability. It even hurts slightly when we randomly "self-permute" S trees to use other word orders that are common in S itself! Presumably this is because the authors of the original S sentences chose, or were required, to order each constituent in a way that would enhance its parsability in context: see the last paragraph of Section 3.3.2.

Synthesizing languages is a balancing act. The synthetic languages are not useful if all of them are too conservatively close to their real sources to add diversity—or too radically different to belong in the galaxy of natural languages. Fortunately, we are at neither extreme. Figure 3.4 visualizes a small sample of 110 languages from our collection.[8] For each ordered pair of languages (S, T), we defined the dissimilarity $d(S, T)$ as the *decrease* in UAS when we parse the dev data of T using a parser trained on S instead of one trained on T. Small dissimilarity (i.e., good parsing transfer) translates to small distance in the figure. The figure shows that the permutations of a substrate language (which share its color) can be radically different from it, as we already saw above. Some may be unnatural, but others are similar to other real languages, including held-out dev languages. Thus Dutch (nl) and Estonian (et) have close synthetic neighbors within this small sample, although they have no close real neighbors. As future work, we want to quantitively compare the synthetic languages with the real ones by measuring some precision and recall errors in the their distibutions (for example, computing the inclusive and exlusice KL-divergeneces on the language model trained on synthetic and real languages).

3.7 An Experiment

We now *illustrate* the use of GD by studying how expanding the set of available treebanks can improve a simple NLP method, related to Figure 3.4.

[8]For each of the 10 real training languages, we sampled 9 synthetic languages: 3 `N`-permuted, 3 `V`-permuted and 3 $\{$`N`,`V`$\}$-permuted. We also included all 10 training + 8 dev languages.

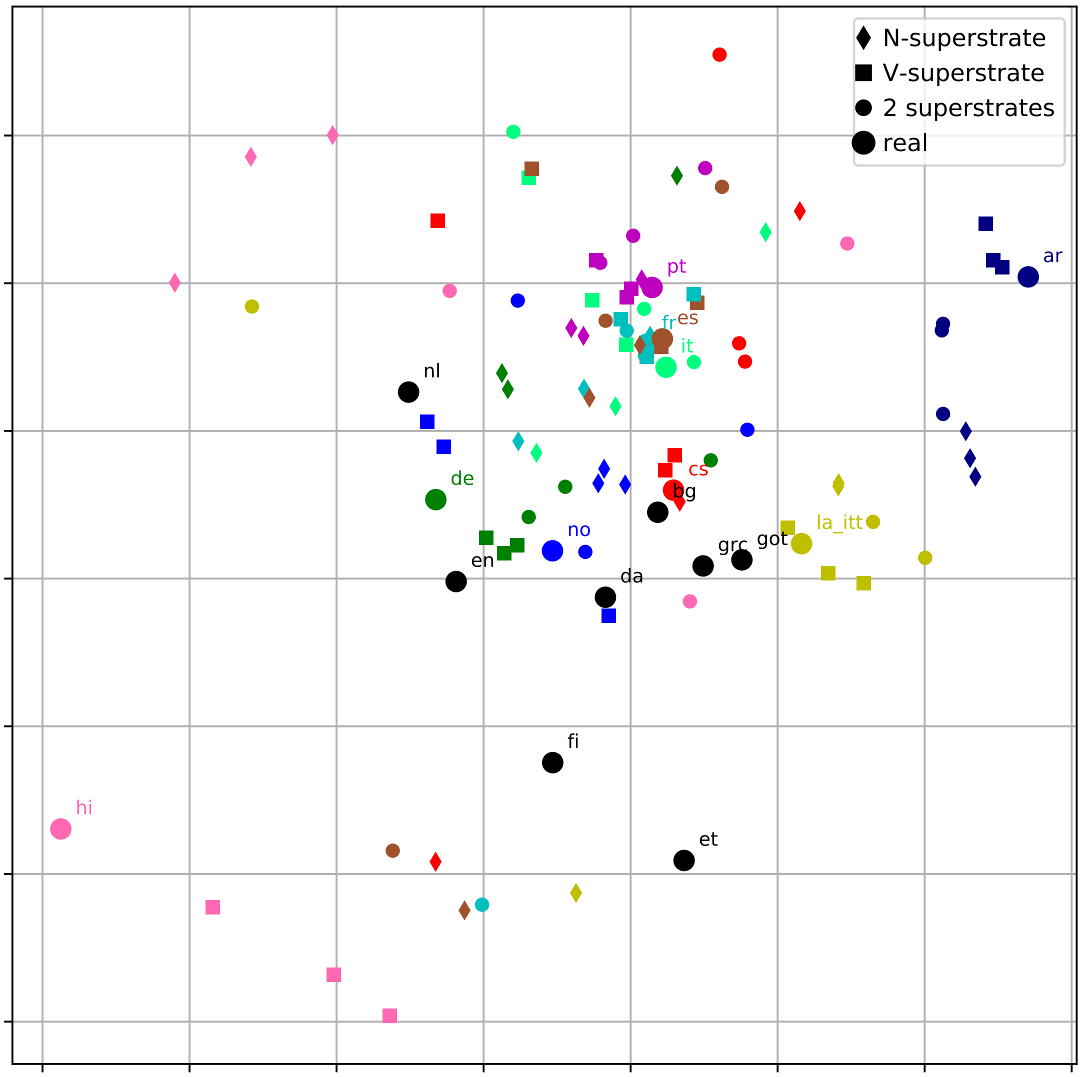


Figure 3.4: Each point represents a language. The color of a synthetic language is the same as its substrate language. Dev languages are shown in black. This 2-dimensional embedding was constructed using metric multidimensional scaling (Borg and Groenen, 2005) on a symmetrized version of our dissimilarity matrix (which is not itself a metric). The embedded distances are reasonably faithful to the symmetrized dissimilarities: metric MDS achieves a low value of 0.20 on its "stress" objective, and we find that Kendall's tau = 0.76, meaning that if one pair of languages is displayed as farther apart than another, then in over 7/8 of cases, that pair is in fact more dissimilar. Among the real languages, note the clustering of Italic languages (pt, es, fr, it), Germanic languages (de, no, en, nl, da), Slavic languages (cs, bg), and Uralic languages (et, fi). Outliers are Arabic (ar), the only Afroasiatic language here, and Hindi (hi), the only SOV language, whose permutations are less outré than it is.

3.7.1 Single-source selection

We use a simple method called "single-source selection": parsing a target language T with a parser that was trained on a source language S, where the two languages are syntactically similar. Such single-source selection parsers are not state-of-the-art (Ganchev et al., 2010; McDonald, Petrov, and Hall, 2011; Ma and Xia, 2014; Rosa and Žabokrtský, 2015a; Rosa and Žabokrtský, 2015b; Guo et al., 2015; Duong et al., 2015b; Rasooli and Collins, 2015), but they have shown substantial improvements over fully unsupervised grammar induction systems (Klein and Manning, 2004; Smith and Eisner, 2006; Spitkovsky, Alshawi, and Jurafsky, 2013).

It is permitted for S and T to have different vocabularies. The S parser can nonetheless parse T (as in Figure 3.4)—provided that it is a "delexicalized" parser that only cares about the POS tags of the input words. In this case, we require only that the target sentences have already been POS tagged using the same tagset as S: in our case, the UD tagset.

3.7.2 Experimental setup

We evaluate single-source selection when the pool of m source languages consists of n real UD languages, plus $m-n$ synthetic GD languages derived by "remixing" just these real languages.[9] We try various values of n and m, where n can be as large as 10 (training languages from Table 3.3) and m can be as large as $n \times (n+1) \times (n+1) \leq 1210$ (see Section 3.5).

Given a *real* target language T from outside the pool, we *select* a single source

[9]The $m-n$ GD treebanks are comparatively impoverished because—in the current GD release—they include only projective sentences (Table 3.2). The n UD treebanks are unfiltered.

language S from the pool, and try to parse UD sentences of T with a parser trained on S. We evaluate the results on T by measuring the unlabeled attachment score (UAS), that is, the fraction of word tokens that were assigned their correct parent. In these experiments (unlike those of Section 3.6), we always evaluate fairly on T's *full* dev or test set from UD—not just the sentences we kept for its GD version (cf. Table 3.2).[10]

The hope is that a large pool will contain at least one language—real or synthetic—that is "close" to T. We have two ways of trying to select a source S with this property:

Supervised selection selects the S whose parser achieves the highest UAS on 100 training sentences of language T. This requires 100 good trees for T, which could be obtained with a modest investment—a single annotator attempting to follow the UD annotation standards in a consistent way on 100 sentences of T, without writing out formal T-specific guidelines. (There is no guarantee that selecting a parser on *training* data will choose well for the *test* sentences of T. We are using a small amount of data to select among *many* dubious parsers, many of which achieve similar results on the training sentences of T. Furthermore, in the UD treebanks, the test sentences of T are sometimes drawn from a different distribution than the training sentences.)

Unsupervised selection selects the S whose training sentences had the best "coverage" of the POS tag sequences in the actual data from T that we aim to parse. More precisely, we choose the S that maximizes $p_S(\text{tag sequences from } T)$—in other words, the maximum-likelihood S—where p_S is our trigram language model for the tag sequences of S. This approach is similar to Rosa and Žabokrtský (2015a), except that they select S based on the fractional counts of the trigrams instead.

[10]The Yara parser can only produce projective parses. It attempts to parse all test sentences of T projectively, but sadly ignores non-projective training sentences of S (as can occur for real S).

Algorithm 5 The algorithm of data collection for one graph, the mean lines in the "kite graph" (Figure 3.5) are actually obtained by averaging 10,000 graphs. Each of these graphs is "smooth" because it incrementally adds new languages as n or m increases. All random choices are made uniformly.

```
Input: Sets $\mathcal{T}$ (target languages), $\mathcal{S}$ (real source languages), $\mathcal{S}'$ (synthetic source languages)
Output: Sets of data points $D_{\text{sup}}$, $D_{\text{unsup}}$
 1: procedure CollectData
 2:     $D \leftarrow \varnothing$
 3:     Sample a target language $T$ from $\mathcal{T}$
 4:     $L \leftarrow$ random.shuffle$(\mathcal{S} - \{T\})$
 5:     $L' \leftarrow$ random.shuffle$(\mathcal{S}')$
 6:     for $n = 1$ to $|L|$ do
 7:         $L'' \leftarrow$ a filtered version of $L'$ that excludes languages with substrates or superstrates
                outside $\{L_1, \ldots, L_n\}$
 8:         for $n' = 1$ to $|L''|$ do
 9:             $\mathcal{P} \leftarrow \{L_1, \ldots, L_n, L''_1, \ldots, L''_{n'}\}$
10:             $m \leftarrow |\mathcal{P}|$
11:             $D_{\text{sup}} \leftarrow D_{\text{sup}} \cup \{(n, m, \text{UAS}_{\text{sup}}(\mathcal{P}, T))\}$   ▷ Add the UAS using the
        supervised selection from $\mathcal{P}$
12:             $D_{\text{unsup}} \leftarrow D_{\text{unsup}} \cup \{(n, m, \text{UAS}_{\text{unsup}}(\mathcal{P}, T))\}$   ▷ Add the UAS using the
        unsupervised selection from $\mathcal{P}$
13:     return $(D_{\text{sup}}, D_{\text{unsup}})$
```

3.7.3 Results

Our most complete visualization is Figure 3.5, which we like to call the "kite graph" for its appearance. We plot the UAS on the development treebank of T as a function of n, m, and the selection method. As Algorithm 5 details, each point on this graph is actually an average over 10,000 experiments that make random choices of T (from the UD development languages), the n real languages (from the UD training languages), and the $m - n$ synthetic languages (from the GD languages derived from the n real languages). We see from the black lines that increasing the number of real languages n is most beneficial. But crucially, when n is fixed in practice, gradually increasing m

by remixing the real languages does lead to meaningful improvements. This is true for both selection methods. As shown in Table 3.4, supervised selection is markedly better than unsupervised.

The "selection graph" in Figure 3.6 visualizes the same experiments in a different way. Here we ask about the fraction of experiments in which using the full pool of m source languages was strictly better than using only the n real languages. We find that when m has increased to its maximum, the full pool nearly always contains a synthetic source language that gets better results than anything in the real pool. After all, our generation of "random" languages is a scattershot attempt to hit the target: the more languages we generate, the higher our chances of coming close. However, our selection methods only manage to *pick* a better language in about 60% of those experiments.

Figure 3.7 offers a fine-grained look at which real and synthetic source languages S succeeded best when $T =$ English. Each curve shows a different superstrate, with the x-axis ranging over substrates. (The figure omits the hundreds of synthetic source languages that use two distinct superstrates, $R_{\mathrm{V}} \neq R_{\mathrm{N}}$.) Real languages are shown as solid black dots, and are often beaten by synthetic languages. For comparison, this graph also plots results that "cheat" by using English supervision.

The above graphs are evaluated on development sentences in development languages. For our final results, Table 3.4, we finally allow ourselves to try transferring to the UD test languages, and we evaluate on test sentences. The comparison is similar to the comparison in the selection graph: do the synthetic treebanks add value? We use our largest source pools, $n = 10$ and $m = 1210$. With supervised selection, selecting the source language from the full pool of m options (not just the n real languages)

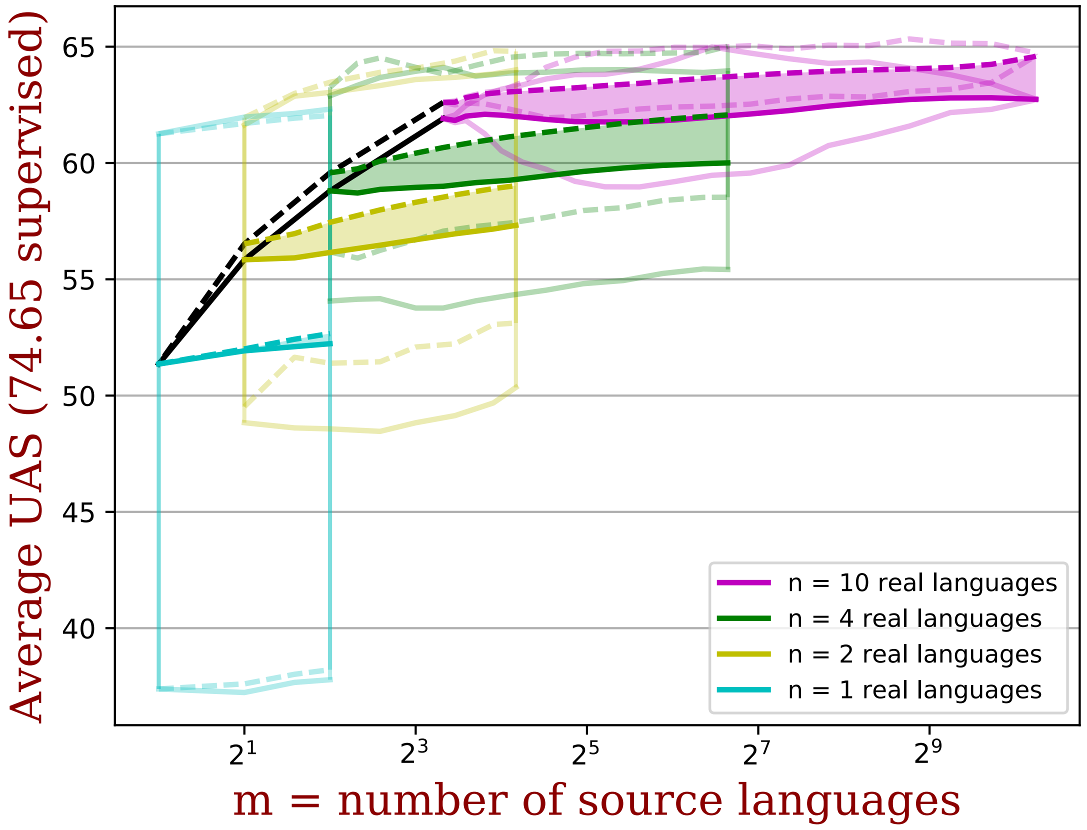


Figure 3.5: Comprehensive results for single-source selection from a pool of m languages (the horizontal axis) synthesized from n real languages. For each color $1, 2, \ldots, n$, the upper dashed line shows the UAS achieved by supervised selection; the lower solid line shows unsupervised selection; and the shaded area highlights the difference. The black dashed and solid lines connect the points where $m = n$, showing how rapidly UAS increases with n when only real languages are used.

Each point is the *mean* dev UAS over 10,000 experiments. We use paler lines in the same color and style to show the considerable *variance* of these UAS scores. These essentially delimit the interdecile range from the 10th to the 90th percentile of UAS score. However, if the plot shows a mean of 57, an interdecile range from 53 to 61 actually means that the middle 80% of experiments were within ± 4 percentage points of the mean UAS *for their target language*. (In other words, before computing this range, we adjust each UAS score for target T by subtracting the mean UAS from the experiments with target T, and adding back the mean UAS from all 10,000 experiments (e.g., 57).)

Notice that on the $n = 10$ curve, there is no variation among experiments either at the minimum m (where the pool always consists of all 10 real languages) or at the maximum m (where the pool always consists of all 1210 galactic languages).

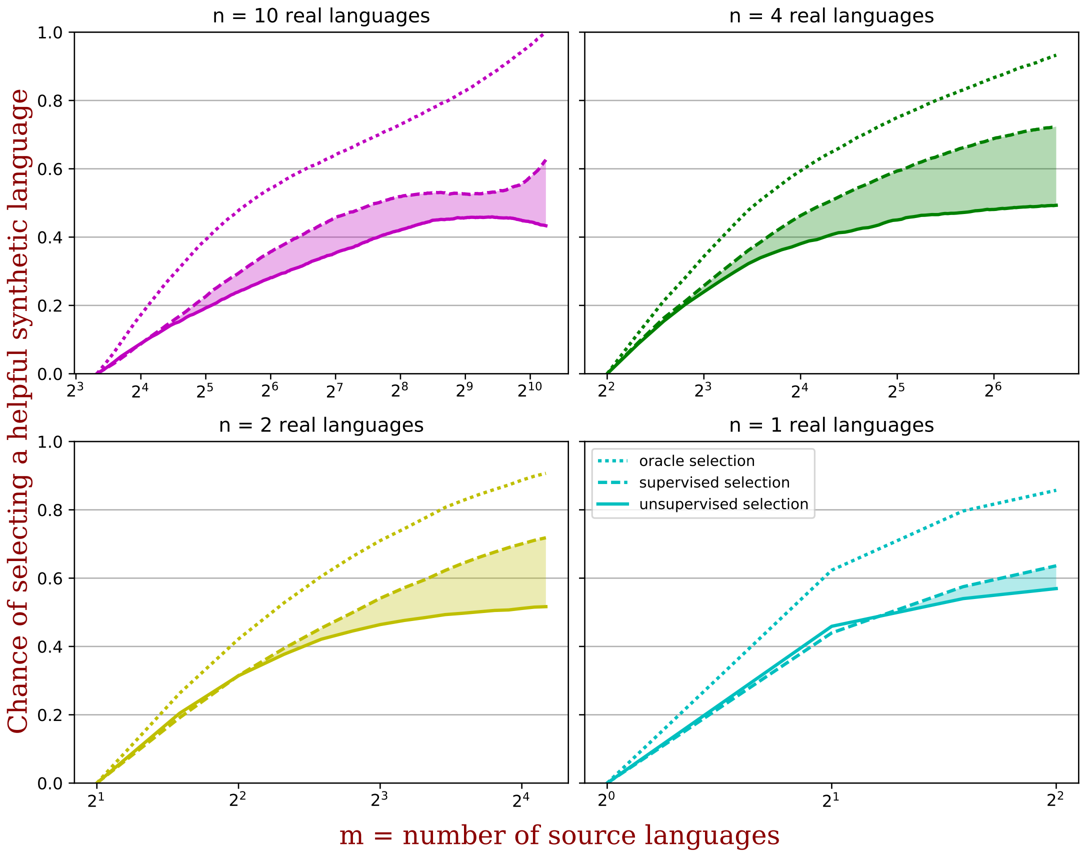


Figure 3.6: Chance that selecting a source from m languages achieves strictly better dev UAS than just selecting from the n real languages.

tends to achieve significantly better UAS on the target language, often dramatically so. On average, the UAS on the test languages increases by 2.3 percentage points, and this increase is statistically significant across these 15 data points. With unsupervised selection, UAS increases by 0.74 points on average, but this difference could be a chance effect.

3.7.4 Experiment with Noisy Tags

The results above use gold POS tag sequences for T. These may not be available if T is a low-resource language. Table 3.5 repeats the single-source selection experiment

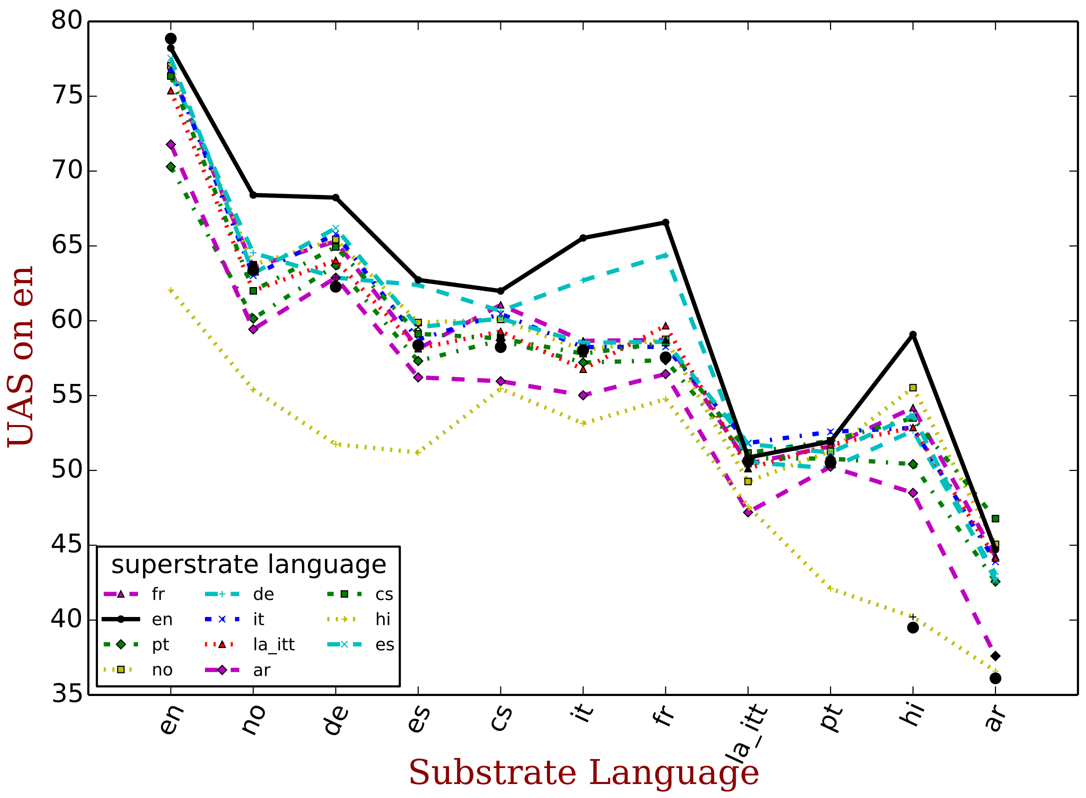


Figure 3.7: UAS performance of different source parsers when applied to English development sentences. The x axis shows the 10 real training languages S, in decreasing order of their UAS performance (plotted as large black dots). For each superstrate R, we plot a curve showing—for each substrate S—the best UAS of the languages $S[R/\mathtt{N}], S[R/\mathtt{V}]$ and $S[R/\mathtt{N}, R/\mathtt{V}]$. The points where $R = S$ are specially colored in black; these are instances of *self-permutation* (Section 3.5). We also add "cheating results" where English itself is used as the substrate (left column) and/or the superstrate (solid black line at top). Thus, the large black dot at the upper left is a supervised English parser.

using noisy automatic POS tags for T for both parser input and unsupervised selection. We obtained the tags using RDRPOSTagger (Nguyen et al., 2014) trained on just 100 gold-tagged sentences (the same set used for supervised selection). The low tagging accuracy does considerably degrade UAS and muddies the usefulness of the synthetic sources.

	unsupervised		(weakly) supervised	
target	real	+synthetic	real	+synthetic
Basque	47.12	**48.97**	45.35	**52.90**
Croatian	**68.69**	**68.89**	**68.69**	**69.11**
Greek	60.07	**65.72**	**65.87**	**66.98**
Hebrew	**63.39**	60.65	62.86	**64.28**
Hungarian	56.41	**64.67**	56.72	**66.22**
Indonesian	**63.79**	61.89	65.36	65.36
Irish	53.55	**59.38**	57.72	**64.72**
Japanese	**62.51**	54.04	**62.51**	**62.49**
Old Church Slavonic	54.11	**57.89**	54.11	**59.28**
Persian	53.41	**58.37**	53.41	**60.18**
Polish	**75.69**	**74.63**	**75.69**	73.05
Romanian	**66.33**	**68.01**	**71.38**	**69.19**
Slovenian	80.41	80.41	80.41	80.41
Swedish	74.96	74.96	74.96	74.96
Tamil	**63.15**	56.20	63.15	63.15
Test Avg.	**62.91**	**63.65**	63.88	**66.15**
Ancient Greek (Avg.)	**46.68**	**46.94**	49.10	**50.95**
Bulgarian	**79.80**	74.52	79.80	79.80
Danish	71.65	71.65	**71.65**	**70.79**
Dutch	**58.44**	**57.94**	**58.44**	**57.85**
Estonian	**68.83**	**72.21**	68.83	**74.75**
English	**63.37**	61.37	63.37	**65.43**
Finnish	51.28	**55.21**	**54.46**	**55.21**
Gothic	54.98	**57.57**	54.98	**58.66**
All Avg.	**62.55**	**63.13**	63.43	**65.47**

Table 3.4: Our final comparison on the 15 test languages appears in the upper part of this table. We ask whether single-source selection to these 15 real target languages is improved by augmenting the source pool of 10 real languages with 1200 synthetic languages. When different languages are selected in these two settings, we boldface the setting with higher test UAS, or both settings if they are not significantly different (paired permutation test by sentence, $p < 0.05$). For completeness, we extend the table with the 10 development languages. The "Avg." lines report the average of 15 test or 23 test+dev languages. The two supervised-selection averages are significantly different (paired permutation test by language, $p < 0.05$).

	tag	unsupervised		(weakly) superv.	
Language		real	+synth	real	+synth
Ancient_Greek (Avg.)	70.87	**37.31**	36.77	**38.51**	**39.28**
Bulgarian	78.33	53.24	**55.08**	53.24	53.24
Danish	78.04	**47.98**	43.40	**47.98**	45.89
Dutch	71.70	**39.40**	**38.99**	**42.42**	**42.75**
Estonian	72.88	45.19	**54.81**	**56.07**	**55.09**
English	77.33	**48.29**	44.40	**48.29**	**48.15**
Finnish	65.65	**29.59**	**28.81**	**36.85**	**36.90**
Gothic	76.66	**44.77**	**44.05**	44.77	**46.83**
Avg.	73.93	**43.22**	**43.29**	**46.02**	**46.02**

Table 3.5: Tagging accuracy on the 8 dev languages, and UAS of the selected source parser with these noisy target-language tag sequences. The results are formatted as in Table 3.4, but here all results are on dev sentences.

3.7.5 Discussion

Many of the curves in Figure 3.5–3.6 still seem to be increasing steadily at maximum m, which suggests that we would benefit from finding ways to generate even more synthetic languages. Diversity of languages seems to be crucial, since adding new real languages improves performance much faster than remixing existing languages. This suggests that we should explore making more extensive changes to the UD treebanks (see Section 3.8).

Surprisingly, Figure 3.5–3.6 show improvements even when $n = 1$. Evidently, self-permutation of a single language introduces some useful variety, perhaps by transporting specialized word orders (e.g., English still allows some limited V2 constructions) into contexts where the source language would not ordinarily allow them but the target language does.

Figure 3.5 shows why unsupervised selection is considerably worse on average than supervised selection. Its 90th percentile is comparable, but at the 10th percentile—presumably representing experiments where no good sources are available—the unsupervised heuristic has more trouble at choosing among the mediocre options. The supervised method can actually test these options using the true loss function.

Figure 3.7 is interesting to inspect. English is essentially a Germanic language with French influence due to the Norman conquest, so it is reassuring that German and French substrates can each be improved by using the other as a superstrate. We also see that Arabic and Hindi are the worst source languages for English, but that Hindi[Arabic/V] is considerably better. This is because Hindi is reasonably similar to English once we correct its SOV word order to SVO by interpolating (Equation (3.3)) with the parameter vector of Arabic, which is VSO.

3.8 Conclusions and Future Work

This work may unlock a wide variety of research opportunities (discussed in Section 3.1). Our empirical studies show that the synthetic languages in this collection remain somewhat natural while improving the diversity of the collection. As a simplistic but illustrative use of the resource, we carefully evaluated its impact on the naive technique of single-source selection parsing. We found that performance could consistently be improved by adding synthetic languages to the pool of sources, assuming gold POS tags.

There are several non-trivial opportunities for improving and extending our treebank collection in future releases.

1. Our current method is fairly conservative, only synthesizing languages with word orders already attested in our small collection of real languages. This does not

increase the diversity of the pool as much as when we add new real languages. Thus, we are particularly interested in generating a wider range of synthetic languages. We could condition reorderings on the surrounding tree structure, as noted in Section 3.3.2. We could choose reordering parameters $\boldsymbol{\theta}_A$ more adventurously than by drawing them from a single known superstrate language. We could go beyond reordering, to systematically choose what function words (determiners, prepositions, particles), function morphemes, or punctuation symbols[11] should appear in the synthetic tree, or to otherwise alter the structure of the tree (Dorr, 1993). In machine translation, researchers (Yamada and Knight, 2001; Eisner, 2003; Galley et al., 2004) have found these additional operations to be crucial for transducing constituency trees or functional-heading (UD's design is content-heading) dependency trees from source languages to target languages. Although these options may produce implausible languages, we could mitigate this by filtering or reweighting our sample of synthetic languages—via rejection sampling or importance sampling—so that they are distributed more like real languages, as measured by their parsabilities, dependency lengths, and estimated WALS features (Dryer and Haspelmath, 2013).

2. Currently, our reordering method only generates *projective* dependency trees.

[11] Our current handling of punctuation produces unnatural results, and not merely because we treat all tokens with tag PUNCT as interchangeable. Proper handling of punctuation and capitalization would require more than just reordering. For example, "`Jane loves her dog, Lexie.`" should reorder into "`Her dog, Lexie, Jane loves.`", which has an extra comma and an extra capital. Accomplishing this would require first recovering a richer tree for the original sentence, in which the appositive `Lexie` is bracketed by a *pair* of commas and the name `Jane` is *doubly* capitalized. These extra tokens were not apparent in the original sentence's surface form because the final comma was absorbed into the adjacent period, and the start-of-sentence capitalization was absorbed into the intrinsic capitalization of `Jane` (Nunberg, 1990). The tokenization provided by the UD treebanks unfortunately does not attempt to undo these orthographic processes, even though it undoes some morphological processes such as contraction. Our later work (Li, Wang, and Eisner, 2019) proposes a generative model which treats the punctuation as bindings of the dependency relations during the permutation process.

We should extend it to allow non-projective trees as well—for example, by pseudo-projectivizing the substrate treebank (Nivre and Nilsson, 2005) and then deprojectivizing it after reordering. One challenge about this approach is that the reordering model should also refine some pseudo dependency labels along the permutation, which is invariant for the projective reordering.

3. The treebanks of real languages can typically be augmented with larger unannotated corpora in those languages (Majliš, 2011), which can be used to train word embeddings and language models, and can also be used for self-training and bootstrapping methods. We plan to release comparable unannotated corpora for our synthetic languages, by automatically parsing and permuting the unnanotated corpora of their substrate languages.

4. At present, all languages derived from an English substrate use the English vocabulary. In the future, we plan to encipher that vocabulary separately for each synthetic language, perhaps choosing a cipher so that the result loosely conforms to the realistic phonotactics and/or orthography of some superstrate language. This would let multilingual methods exploit lexical features without danger of overfitting to specific lexical items that appear in many synthetic training languages. Although prefixing a language ID to each word could also do this, alphabetic ciphers can preserve features of words that are potentially informative for linguistic structure discovery: their cooccurrence statistics, their length and phonological shape, and the sharing of substrings among morphologically related words.

5. Finally, this chapter has focused on generating a broadly reusable collection of synthetic treebanks. For some applications (including single-source selection), one might wish to tailor a synthetic language on demand, e.g., starting with one of our

treebanks but modifying it further to more closely match the surface statistics of a given target language (Dorr et al., 2002). Chapter 6 will continue this discussion on how to generate "targeted" synthetic languages.

We conclude by revisiting our opening point of this chapter (as well as Sections 1.4.1 and 2.5). Unsupervised discovery of linguistic structure is difficult. We often do not know quite what function to maximize, or how to globally maximize it. If we could make labeled languages as plentiful as labeled images, then we could treat linguistic structure discovery as a problem of *supervised* prediction—one that need not succeed on all formal languages, but which should generalize at least to the domain of *possible* human languages.

Chapter 4

Fine-Grained Prediction of Syntactic Typology

Chapter 3 introduces a large set of synthetic languages generated by mix-and-match over some real languages. This chapter and the next (Chapter 5) will study how this novel dataset could mitigate the data sparsity issue (Section 2.5) of training a good amortized Bayes (AB) inference function (Section 2.4.4). While Chapter 5 focuses on predicting unsupervised parsers—our final goal, this chapter shows how to predict the basic word order facts known as *the syntactic typology*. Different from the parsing task, which predicts dependency relations for each token, this chapter makes the prediction at the *dependency type level*. For example, we predict how often direct objects follow their verbs, how often adjectives follow their nouns, and in general the directionalities of all dependency relations. Like unsupervised parsing, this problem is usually regarded as unsupervised learning. We adopt the AB inference function by treating it as supervised learning, using the large collection of Galactic Dependencies (GD) languages (Chapter 3) as training data. The AB inference function must identify *surface* features of a language's POS sequence (hand-engineered or neural features) that correlate with the language's syntactic typology.

To some extent, this task could be considered as a preliminary to unsupervised parsing because a good parser must know these word order facts in order to generate good parses. In the experiment, we show: 1) Given a small set of real languages, it helps to add many GD languages to the training data, 2) our inference function is robust even when the POS sequences include noise, and 3) our inference function outperforms a grammar induction baseline by a large margin.

4.1 Introduction

Descriptive linguists often characterize a human language by its *typological properties*. For instance, English is an SVO-type language because its basic clause order is Subject-Verb-Object (SVO), and also a prepositional-type language because its adpositions normally precede the noun. Identifying basic word order must happen early in the acquisition of syntax, and presumably guides the initial interpretation of sentences and the acquisition of a finer-grained grammar. Our AB inference function is suitable for doing this.

The problem is challenging because the language's true word order statistics are computed from syntax trees, whereas our method has access only to a POS-tagged corpus. Based on these POS sequences alone, we predict the *directionality* of each type of dependency relation. We define the directionality to be a real number in $[0, 1]$: The fraction of tokens of this relation that are "right-directed," in the sense that the child (modifier) falls to the right of its parent (head). For example, the `dobj` relation points from a verb to its direct object (if any), so a directionality of 0.9—meaning that 90% of `dobj` dependencies are right-directed—indicates a dominant verb-object order. (See Table 4.1 for more such examples.) As discussed in Section 2.4.4, we denote $\boldsymbol{\tau}(\boldsymbol{y})$ to be some typological information that AB inference function is trained to predict. In this chapter, the form of $\boldsymbol{\tau}(\boldsymbol{y})$ is a vector of directionalities of 57 dependency types in the UD project. We assume that all languages draw on the same set of POS tags and dependency relations that is proposed by the UD project (see Section 3.3.1), so that our predictor works across languages.

Why do this? Liu (2010) has argued for using these directionality numbers in $[0,1]$ as fine-grained and robust *typological descriptors*. Besides the linguistic merit on its own, we believe that the directionalities could also be used to help define an *initializer, prior, or regularizer* for NLP tasks like grammar induction or syntax-based machine translation. Finally, the $\boldsymbol{\tau}(\boldsymbol{y})$ vector can be regarded as a *language embedding* computed from the POS-tagged corpus. This language embedding may be useful as an input to multilingual NLP systems, such as the cross-linguistic neural dependency parser of Ammar et al. (2016). In fact, some multilingual NLP systems already condition on typological properties looked up in the World Atlas of Language Structures, or WALS (Dryer and Haspelmath, 2013), as we review in Section 4.7. However, WALS does not list all properties of all languages, and may be somewhat inconsistent since it collects work by many linguists. Also, WALS only gives discrete categories such as SVO and VSO, not fractions. Our system provides an automatic alternative as well as a methodology for generalizing to new properties.

More broadly, this task is motivated by the challenge of determining the structure of a language from its superficial features. Principles & Parameters theory (Chomsky, 1981; Chomsky and Lasnik, 1993) famously—if controversially—hypothesized that human babies are born with an evolutionarily tuned system that is specifically adapted

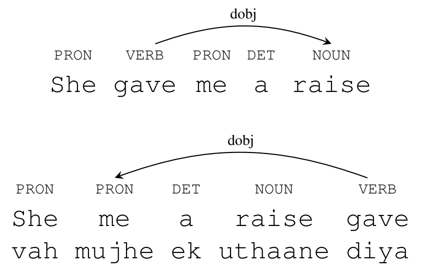
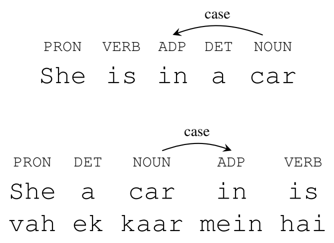
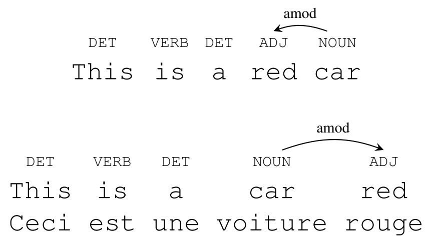
Typology	Example
Verb-Object (English) Object-Verb (Hindi)	 dobj PRON VERB PRON DET NOUN She gave me a raise dobj PRON PRON DET NOUN VERB She me a raise gave vah mujhe ek uthaane diya
Prepositional (English) Postpositional (Hindi)	 case PRON VERB ADP DET NOUN She is in a car case PRON DET NOUN ADP VERB She a car in is vah ek kaar mein hai
Adjective-Noun (English) Noun-Adjective (French)	 amod DET VERB DET ADJ NOUN This is a red car amod DET VERB DET NOUN ADJ This is a car red Ceci est une voiture rouge

Table 4.1: Three typological properties in the World Atlas of Language Structures (Dryer and Haspelmath, 2013), and how they affect the directionality of Universal Dependencies relations.

to natural language, which can predict typological properties ("parameters") by spotting telltale configurations in purely linguistic input. Here we investigate whether such a system can be tuned by gradient descent. It is at least plausible that useful superficial features do exist: e.g., if nouns often precede verbs but rarely follow verbs, then the language may be verb-final.

4.2 Approach

We depart by revisiting the discussion in Sections 1.3 and 2.5 on the difficulties of the traditional approach to latent structure discovery, namely unsupervised learning. Unsupervised syntax learners in NLP tend to be rather inaccurate—partly because they are failing to maximize an objective that has many local optima (the search error), and partly because that objective does not capture all the factors that linguists consider when assigning syntactic structure (the model error).

The idea of AB inference function is a supervised approach, where the heuristic is simply imitate how linguists have analyzed other languages. This meta-supervised objective goes beyond the log-likelihood of a PCFG-like model given the corpus, because linguists do not merely try to predict the surface corpus. Their dependency annotations may reflect a cross-linguistic theory that considers semantic interpretability and equivalence, rare but informative phenomena, consistency across languages, a prior across languages, and linguistic conventions (including the choice of latent labels such as `dobj`). Our learner does not consider these factors explicitly, but we hope it will identify correlates (e.g., using deep learning) that can make similar predictions.

Being supervised, our objective should also suffer less from local optima. Indeed, we could even set up our problem with a *convex* objective, such as (kernel) logistic regression, to predict each directionality separately. As discussed in Section 2.5, our setting presents unusually sparse data for supervised learning, since *each training example is an entire language*. The world presumably does not offer enough natural languages—particularly with machine-readable corpora—to train a good classifier to detect, say, Object-Verb-Subject (OVS) languages, especially given the class imbalance problem that OVS languages are empirically rare, and the non-IID problem that the available OVS languages may be evolutionarily related.[1] We mitigate this issue by training on the Galactic Dependencies (GD) treebanks (Chapter 3), a collection of more than 50,000 human-like synthetic languages. The treebank of each synthetic language is generated by stochastically permuting the subtrees in a given real treebank to match the word order of other real languages. Thus, we have many synthetic languages that are Object-Verb like Hindi but also Noun-Adjective like French. We know the true directionality of each synthetic language and we would like our classifier to predict that directionality, just as it would for a real language. We will show that our system's accuracy benefits from fleshing out the training set in this way, which can be seen as a form of regularization.

A possible criticism of our work is that obtaining the input POS sequences requires human annotators, and perhaps these annotators could have answered the typological classification questions as well. Arguably, this criticism also applies to most work on grammar induction. We will show that our system is at least robust to noise in the input POS sequences (Section 4.6.4).

4.3 Task Formulation

We now formalize the setup of the fine-grained typological prediction task under the framework of Section 2.4.4. Let $\mathcal{R}$ be the set of universal relation types.[2] We use $\xrightarrow{r}$ to denote a rightward dependency token with label $r \in \mathcal{R}$.

Input for language ℓ: A POS-tagged corpus $\boldsymbol{x}$.

Output for language ℓ: Our system predicts $\hat{p}(\rightarrow \mid r, \ell)$, the probability that a

[1]Properties shared within an OVS language family may appear to be consistently predictive of OVS, but are actually confounds that will not generalize to other families in test data.

[2]In our final evaluation $\mathcal{R}$ has 57 relation types. See Section 4.6.9 for a detailed discussion.

token in language ℓ of an r-labeled dependency will be right-oriented. It predicts this for *each* dependency relation type $r \in \mathcal{R}$, such as $r = \texttt{dobj}$. Thus, the output of $e_{\boldsymbol{\Theta}}(\boldsymbol{x})$ is a vector of predicted probabilities $\hat{\boldsymbol{\tau}}(\boldsymbol{x}) \overset{\text{def}}{=} \hat{\mathbf{p}} \in [0,1]^{|\mathcal{R}|}$.

Training: Following Algorithm 4, we set the parameter $\boldsymbol{\Theta}$ of our system using a collection of training pairs $(\boldsymbol{x}_i, \boldsymbol{\tau}(\boldsymbol{y}_i))$, each of which corresponds to the i^{th} UD or GD training language ℓ. $\boldsymbol{\tau}(\boldsymbol{y}_i)$ in is defined as the true vector of probabilities as empirically estimated from ℓ's treebank.

Evaluation: Over pairs $(\boldsymbol{x}, \boldsymbol{\tau}(\boldsymbol{y}))$ that correspond to held-out *real* languages, we evaluate the expected loss of the prediction $\hat{\boldsymbol{\tau}}_{\text{AB}}(\boldsymbol{x})$. We use ε-insensitive loss[3] with $\varepsilon = 0.1$, so our evaluation metric is

$$\text{Loss}(\hat{\boldsymbol{\tau}}_{\text{AB}}(\boldsymbol{x}), \boldsymbol{\tau}(\boldsymbol{y})) = \sum_{r \in \mathcal{R}} p(r \mid \ell) \cdot \text{loss}_{\varepsilon}(\hat{p}(\rightarrow \mid r, \ell), p(\rightarrow \mid r, \ell)) \tag{4.1}$$

where

- $\text{loss}_{\varepsilon}(\hat{p}, p) \overset{\text{def}}{=} \max(|\hat{p} - p| - \varepsilon, 0)$
- $p(\rightarrow \mid r, \ell) = \frac{\text{count}_{\ell}(\xrightarrow{r})}{\text{count}_{\ell}(r)}$ is the empirical estimate of the directionality of r in ℓ.
- $\hat{p}(\rightarrow \mid r, \ell)$ is the system's prediction

The aggregate metric Equation (4.1) is an expected loss that is weighted by $p(r \mid \ell) = \frac{\text{count}_{\ell}(r)}{\sum_{r' \in \mathcal{R}} \text{count}_{\ell}(r')}$, to emphasize relation types that are more frequent in ℓ.

Why this loss function? We chose an L1-style loss, rather than L2 loss or log-loss, so that the aggregate metric is not dominated by outliers. We took $\varepsilon > 0$ in order to forgive small errors: If some predicted directionality is already "in the ballpark," we prefer to focus on getting other predictions right, rather than fine-tuning this one. Our

[3]Proposed by Drucker et al. (1997), Smola and Schölkopf (2004), and Vapnik (2013) for support vector regression.

intuition is that errors $< \varepsilon$ in $\hat{\tau}_{\text{AB}}(\boldsymbol{x})$'s elements will not greatly harm downstream tasks that analyze individual sentences, and might even be easy to correct by grammar reestimation (e.g., EM) that uses $\hat{\tau}_{\text{AB}}(\boldsymbol{x})$ as a starting point.

In short, we have the intuition that if our predicted $\hat{\tau}_{\text{AB}}(\boldsymbol{x})$ achieves small $\text{loss}_{\varepsilon}$ on the frequent relation types, then $\hat{\tau}_{\text{AB}}(\boldsymbol{x})$ will be helpful for downstream tasks, although testing that intuition is beyond the scope of this work. One could tune ε on a downstream task.

4.4 Simple "Expected Count" Baseline

Before launching into our full models, we warm up with a simple baseline heuristic called *expected count* (EC), which is reminiscent of Principles & Parameters. This baseline doesn't have trainable parameters, thus doesn't need training langauges. The idea is that if `ADJ`s tend to precede nearby `NOUN`s in the sentences of language ℓ, then `amod` probably tends to point leftward in ℓ. After all, the training languages show that when `ADJ` and `NOUN` are nearby, they are usually linked by `amod`.

Fleshing this out, EC estimates directionalities as

$$\hat{p}(\rightarrow \mid r, \ell) = \frac{\text{ecount}_{\ell}(\xrightarrow{r})}{\text{ecount}_{\ell}(\xrightarrow{r}) + \text{ecount}_{\ell}(\xleftarrow{r})} \tag{4.2}$$

where we estimate the expected $\xleftarrow{r}$ and $\xrightarrow{r}$ counts by

$$\text{ecount}_{\ell}(\xrightarrow{r}) = \sum_{x \in \boldsymbol{x}} \sum_{\substack{1 \le i < j \le |x| \\ j - i < w}} p(\xrightarrow{r} \mid x_i, x_j) \tag{4.3}$$

$$\text{ecount}_{\ell}(\xleftarrow{r}) = \sum_{x \in \boldsymbol{x}} \sum_{\substack{1 \le i < j \le |x| \\ j - i < w}} p(\xleftarrow{r} \mid x_i, x_j) \tag{4.4}$$

Here x ranges over tag sequences (sentences) of $\boldsymbol{x}$, and w is a window size that characterizes "nearby."[4]

In other words, we ask: given that x_i and x_j are nearby tag tokens in the test corpus $\boldsymbol{x}$, are they likely to be linked? Equation (4.3)–(4.4) count such a pair as a "soft vote" for $\xrightarrow{r}$ if such pairs tended to be linked by $\xrightarrow{r}$ in the treebanks of the training languages,[5] and a "soft vote" for $\xleftarrow{r}$ if they tended to be linked by $\xleftarrow{r}$.

Training: For any two tag types t, t' in the universal POS tagset $\mathcal{T}$, we simply use the training treebanks to get empirical estimates of $p(\cdot \mid t, t')$, taking

$$p(\xrightarrow{r} \mid t, t') = \frac{\sum_\ell s_\ell \cdot \text{count}_\ell(t \xrightarrow{r} t')}{\sum_\ell s_\ell \cdot \text{count}_\ell(t, t')} \tag{4.5}$$

and similarly for $p(\xleftarrow{r} \mid t, t')$. This can be interpreted as the (unsmoothed) fraction of (t, t') within a w-word window where t is the r-type parent of t', computed by micro-averaging over languages. To get a fair average over languages, Equation (4.5) downweights the languages that have larger treebanks, yielding a *weighted* micro-average in which we define the weight $s_\ell = 1/\sum_{t\in\mathcal{T}, t'\in\mathcal{T}} \text{count}_\ell(t, t')$.

As we report later in Table 4.5, even this simple supervised heuristic performs significantly better than state-of-the-art grammar induction systems. However, it is not a *trained* heuristic: it has no free parameters that we can tune to optimize our evaluation metric. For example, it can pay too much attention to tag pairs that are not discriminative. We therefore proceed to build a trainable, feature-based system.

[4] In our experiment, we chose $w = 8$ by cross-validation over $w = 2, 4, 8, 16, \infty$.

[5] Thus, the EC heuristic examines the correlation between relations and tags in the training treebanks. But our methods in the next section will follow the formalization of Section 4.3: they do not examine a training treebank beyond its directionality vector $\boldsymbol{\tau}(\boldsymbol{y})$.

4.5 Proposed Model Architecture

To train our model, we will try to minimize the evaluation objective Equation (4.1) averaged over the training languages, plus a regularization term given in Section 4.5.4.[6]

4.5.1 Directionality predictions from scores

Our predicted directionality for relation r will be

$$\hat{p}(\rightarrow \mid r, \ell) = 1/(1 + \exp(-\boldsymbol{\psi}(\boldsymbol{x})_r)) \tag{4.6}$$

$\boldsymbol{\psi}(\boldsymbol{x})$ is a parametric function (see Section 4.5.2 below) that maps $\boldsymbol{x}$ to a *score* vector in $\mathbb{R}^{|\mathcal{R}|}$. Relation type r should get positive or negative score according to whether it usually points right or left. The formula above converts each score to a directionality—a probability in $(0, 1)$—using a logistic transform.

4.5.2 Design of the scoring function $\boldsymbol{\psi}(\boldsymbol{x})$

To score all dependency relation types given the corpus $\boldsymbol{x}$, we use a feed-forward neural network with one hidden layer (Figure Figure 4.1):

$$\boldsymbol{\psi}(\boldsymbol{x}) = V\,\sigma(W\boldsymbol{\pi}(\boldsymbol{x}) + \mathbf{b}_W) + \mathbf{b}_V \tag{4.7}$$

$\boldsymbol{\pi}(\boldsymbol{x})$ extracts a d-dimensional feature vector from the corpus $\boldsymbol{x}$ (see Section 4.5.3 below). W is a $h \times d$ matrix that maps $\boldsymbol{\pi}(\boldsymbol{x})$ into a h-dimensional space and $\mathbf{b}_W$ is a h-dimensional bias vector. σ is an element-wise activation function. V is a $|\mathcal{R}| \times h$ matrix whose rows can be regarded as learned embeddings of the dependency relation

[6]We gave all training languages the same weight. In principle, we could have downweighted the synthetic languages as out-of-domain, using cross-validation to tune the weighting.

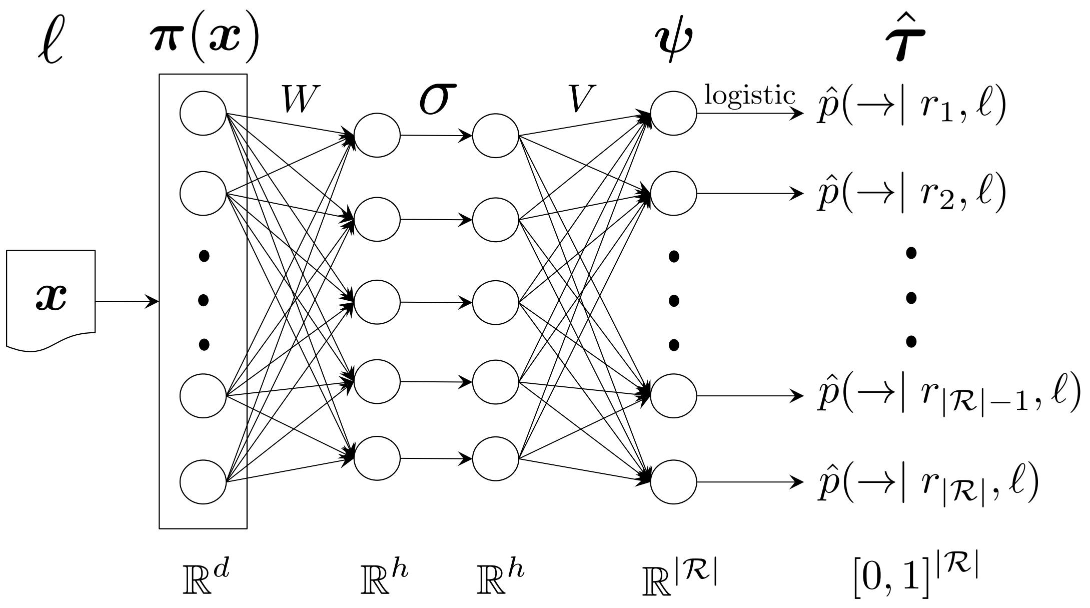

ℓ
$\boldsymbol{\pi}(\boldsymbol{x})$
$\boldsymbol{\psi}$
$\hat{\boldsymbol{\tau}}$
W
$\boldsymbol{\sigma}$
V
logistic
$\hat{p}(\rightarrow \mid r_1, \ell)$
$\hat{p}(\rightarrow \mid r_2, \ell)$
$\hat{p}(\rightarrow \mid r_{|\mathcal{R}|-1}, \ell)$
$\hat{p}(\rightarrow \mid r_{|\mathcal{R}|}, \ell)$
$\boldsymbol{x}$
$\mathbb{R}^d$
$\mathbb{R}^h$
$\mathbb{R}^h$
$\mathbb{R}^{|\mathcal{R}|}$
$[0,1]^{|\mathcal{R}|}$

Figure 4.1: Basic predictive architecture from Equation (4.6)–(4.7). $\mathbf{b}_W$ and $\mathbf{b}_V$ are suppressed for readability.

types. $\mathbf{b}_V$ is a $|\mathcal{R}|$-dimensional bias vector that determines the default rightwardness of each relation type. We give details in Section 4.6.5.

The hidden layer $\sigma(W\boldsymbol{\pi}(\boldsymbol{x}) + \mathbf{b}_W)$ can be regarded as a latent representation of the language's word order properties, from which potentially *correlated* predictions $\hat{\boldsymbol{\tau}}_{\text{AB}}(\boldsymbol{x})$ are extracted.

4.5.3 Design of the featurization function $\boldsymbol{\pi}(\boldsymbol{x})$

Our current feature vector $\boldsymbol{\pi}(\boldsymbol{x})$ considers only the POS tag sequences for the sentences in the unparsed corpus $\boldsymbol{x}$. Each sentence is augmented with a special boundary tag # at the start and end. We explore both hand-engineered features and neural features.

Hand-engineered features. Recall that Section 4.4 considered which tags appeared near one another in a given order. We now devise a slew of features to measure such co-occurrences in a variety of ways. By training the weights of these many features, our system will discover which ones are actually predictive.

Let $g(t \mid j) \in [0,1]$ be some measure (to be defined shortly) of the *prevalence* of tag t near token j of corpus $\boldsymbol{x}$. We can then measure the prevalence of t, both overall and just near tokens of tag s:[7]

$$\pi_t = \operatorname*{mean}_j g(t \mid j) \tag{4.8}$$

$$\pi_{t|s} = \operatorname*{mean}_{j:\, T_j = s} g(t \mid j) \tag{4.9}$$

where T_j denotes the tag of token j. We now define versions of these quantities for particular prevalence measures g.

Given $w > 0$, let the *right window* $\boldsymbol{w}_j$ denote the sequence of tags $T_{j+1}, \ldots, T_{j+w}$ (padding this sequence with additional # symbols if it runs past the end of j's sentence). We define quantities $\pi^w_{t|s}$ and π^w_t via Equation (4.8)–(4.9), using a version of $g(t \mid j)$ that measures the fraction of tags in $\boldsymbol{w}_j$ that equal t. Also, for $b \in \{1, 2\}$, we define $\pi^{w,b}_{t|s}$ and $\pi^{w,b}_t$ using a version of $g(t \mid j)$ that is 1 if $\boldsymbol{w}_j$ contains at least b tokens of t, and 0 otherwise.

For each of these quantities, we also define a corresponding *mirror-image* quantity (denoted by negating $w > 0$) by computing the same feature on a reversed version of the corpus.

We also define "truncated" versions of all quantities above, denoted by writing $\hat{}$ over the w. In these, we use a *truncated window* $\hat{\boldsymbol{w}}_j$, obtained from $\boldsymbol{w}_j$ by removing

[7]In practice, we do backoff smoothing of these means. This avoids subsequent division-by-0 errors if tag t or s has count 0 in the corpus, and it regularizes $\pi_{t|s}/\pi_t$ toward 1 if t or s is rare. Specifically, we augment the denominators by adding λ, while augmenting the numerator in Equation (4.8) by adding $\lambda \cdot \text{mean}_{j,t}\, g(t \mid j)$ (unsmoothed) and the numerator in Equation (4.9) by adding λ times the smoothed π_t from Equation (4.8). $\lambda > 0$ is a hyperparameter (see Section 4.6.5).

any suffix that starts with # or with a copy of tag T_j (that is, s).[8] As an example, $\pi^{\hat{8},2}_{\mathrm{N}|\mathrm{V}}$ asks how often a verb is followed by at least 2 nouns, within the next 8 words of the sentence *and before the next verb*. A high value of this is a plausible indicator of a VSO-type or VOS-type language.

We include the following features for each tag pair s, t and each $w \in \{1, 3, 8, 100, -1, -3, -8, -100, \hat{1}, \hat{3}, \hat{8}, \hat{100}, -\hat{1}, -\hat{3}, -\hat{8}, -\hat{100}\}$:[9]

$$\pi^w_t,\ \pi^w_{t|s},\ \pi^w_{t|s} \cdot \pi^w_s,\ \pi^w_{t|s} // \pi^w_t,\ \pi^w_t // \pi^w_{t|s},\ \pi^w_{t|s} // \pi^{-w}_{t|s}$$

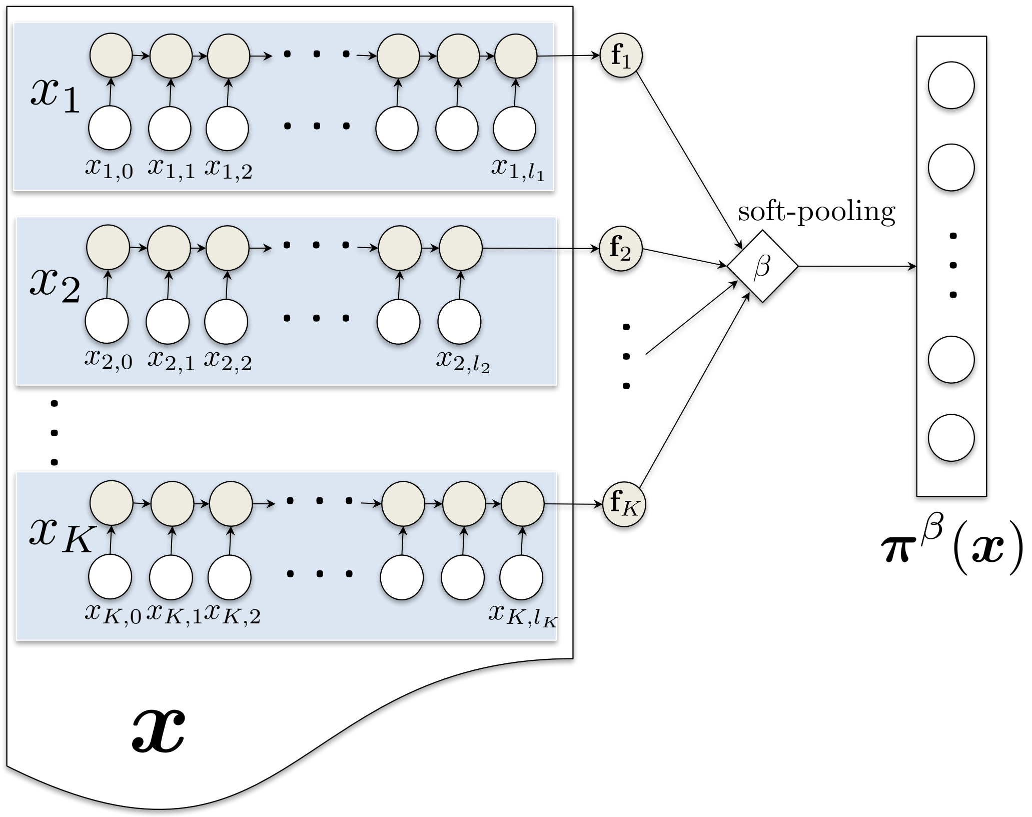
where we define $x//y = \min(x/y, 1)$ to prevent unbounded feature values, which can result in poor generalization. Notice that for $w = 1$, $\pi^w_{t|s}$ is bigram conditional probability, $\pi^w_{t|s} \cdot \pi^w_s$ is bigram joint probability, the log of $\pi^w_{t|s}/\pi^w_t$ is bigram pointwise mutual information, and $\pi^w_{t|s}/\pi^{-w}_{t|s}$ measures how much more prevalent t is to the right of s than to the left. By also allowing other values of w, we generalize these features. Finally, our model also uses versions of these features for each $b \in 1, 2$.

Neural features. As an alternative to the manually designed $\boldsymbol{\pi}$ function above, we consider a neural approach to detect predictive configurations in the sentences of $\boldsymbol{x}$, potentially including complex long-distance configurations. Linguists working with Principles & Parameters theory have supposed that a single telltale sentence—a *trigger*—may be enough to determine a typological parameter, at least given the settings of other parameters (Gibson and Wexler, 1994; Frank and Kapur, 1996).

We map each corpus sentence u_i to a finite-dimensional real vector $\mathbf{f}_i$ by using a gated recurrent unit (GRU) network (Cho et al., 2014b), a type of recurrent neural

[8]In the "fraction of tags" features, $g(t \mid j)$ is undefined ($\frac{0}{0}$) when $\hat{\boldsymbol{w}}_j$ is empty. We omit undefined values from the means.

[9]The reason we don't include $\pi^{-w}_{t|s} // \pi^w_{t|s}$ is that it is included when computing features for $-w$.



x_1
$x_{1,0}$ $x_{1,1}$ $x_{1,2}$ x_{1,l_1}
x_2
$x_{2,0}$ $x_{2,1}$ $x_{2,2}$ x_{2,l_2}
x_K
$x_{K,0}$ $x_{K,1}$ $x_{K,2}$ x_{K,l_K}
$\boldsymbol{x}$
$\mathbf{f}_1$
$\mathbf{f}_2$
$\mathbf{f}_K$
soft-pooling
β
$\boldsymbol{\pi}^{\beta}(\boldsymbol{x})$

Figure 4.2: Extracting and pooling the neural features.

network that is a simplified variant of an LSTM network (Hochreiter and Schmidhuber, 1997). The GRU reads the sequence of one-hot embeddings of the tags in x_i (including the boundary symbols #). We omit the part of the GRU that computes an output sequence, simply taking $\mathbf{f}_i$ to be the final hidden state vector. The parameters are trained jointly with the rest of our typology prediction system, so the training procedure attempts to discover predictively useful configurations.

The various elements of $\mathbf{f}_i$ attempt to detect various interesting configurations in sentence x_i. Some might be triggers (which call for max-pooling over sentences); others might provide softer evidence (which calls for mean-pooling). For generality, therefore, we define our feature vector $\boldsymbol{\pi}(\boldsymbol{x})$ by *soft-pooling* of the sentence vectors

$\mathbf{f}_i$ (Figure 4.2). The tanh gate in the GRU implies $f_{ik} \in (-1, 1)$ and we transform this to the positive quantity $f'_{ik} = \frac{f_{ik}+1}{2} \in (0, 1)$. Given an "inverse temperature" β, define[10]

$$\pi_k^\beta = \left(\underset{i}{\text{mean}} \; (f'_{ik})^\beta \right)^{1/\beta} \tag{4.10}$$

This π_k^β is a pooled version of f'_{ik}, ranging from max-pooling as $\beta \to -\infty$ (i.e., does f'_{ik} fire strongly on any sentence i?) to min-pooling as $\beta \to -\infty$. It passes through arithmetic mean at $\beta = 1$ (i.e., how strongly does f'_{ik} fire on the average sentence i?), geometric mean as $\beta \to 0$ (this may be regarded as an arithmetic mean in log space), and harmonic mean at $\beta = -1$ (an arithmetic mean in reciprocal space).

Our final $\boldsymbol{\pi}$ is a concatenation of the $\boldsymbol{\pi}^\beta$ vectors for $\beta \in \{-4, -2, -1, 0, 1, 2, 4\}$. We chose these β values experimentally, using cross-validation.

Combined model. We also consider a model

$$\boldsymbol{\psi}(\boldsymbol{x}) = \alpha \, \boldsymbol{\psi}_{\mathrm{H}}(\boldsymbol{x}) + (1 - \alpha) \, \boldsymbol{\psi}_{\mathrm{N}}(\boldsymbol{x}) \tag{4.11}$$

where $\boldsymbol{\psi}_{\mathrm{H}}(\boldsymbol{x})$ is the score assigned by the hand-feature system, $\boldsymbol{\psi}_{\mathrm{N}}(\boldsymbol{x})$ is the score assigned by the neural-feature system, and $\alpha \in [0, 1]$ is a hyperparameter to balance the two. $\boldsymbol{\psi}_{\mathrm{H}}(\boldsymbol{x})$ and $\boldsymbol{\psi}_{\mathrm{N}}(\boldsymbol{x})$ were trained separately. At test time, we use Equation (4.11) to combine them linearly before the logistic transform Equation (4.6). This yields a weighted-product-of-experts model.

[10]For efficiency, we restrict the mean to $i \leq 1e4$ (the first 10,000 sentences).

4.5.4 Training procedure

Length thresholding. By default, our feature vector $\pi(x)$ is extracted from those sentences in x with length ≤ 40 tokens. In Section 4.6.3, however, we try concatenating this feature vector with one that is extracted in the same way from just sentences with length ≤ 10. The intuition (Spitkovsky, Alshawi, and Jurafsky, 2010) is that the basic word order of the language can be most easily discerned from short, simple sentences.

Initialization. We initialize the model of Equation (4.6)–(4.7) so that the estimated directionality $\hat{p}(\rightarrow \mid r, \ell)$, regardless of ℓ, is initially a weighted mean of r's directionalities in the training languages, namely

$$\bar{p}_r \stackrel{\text{def}}{=} \sum_{\ell} \mathrm{w}_{\ell}(r)\, p(\rightarrow \mid r, \ell) \tag{4.12}$$

$$\text{where } \mathrm{w}_{\ell}(r) \stackrel{\text{def}}{=} \frac{p(r|\ell)}{\sum_{\ell'} p(r|\ell')} \tag{4.13}$$

This is done by setting $V = 0$ and the bias $(\mathbf{b}_V)_r = \log \frac{\bar{p}_r}{1-\bar{p}_r}$, clipped to the range $[-10, 10]$. As a result, we make sensible initial predictions even for rare relations r, which allows us to converge reasonably quickly even though we do not update the parameters for rare relations as often.

We initialize the recurrent connections in the GRU to random orthogonal matrices. All other weight matrices in Figure 4.1 and the GRU use "Xavier initialization" (Glorot and Bengio, 2010). All other bias weight vectors are initialized to 0.

Regularization. We add an L2 regularizer to the objective. When training the neural network, we use dropout as well. All hyperparameters (regularization coefficient, dropout rate, etc.) are tuned via cross-validation; see Section 4.6.5.

Optimization. We use different algorithms in different feature settings. With scoring functions that use only hand features, we adjust the feature weights by stochastic gradient descent (SGD). With scoring functions that include neural features, we use RMSProp (Tieleman and Hinton, 2012).

4.6 Experiments

Our controlled experiments are conducted by controlling the training languages between the UD v1.2 dataset and its synthetic counterpart, which is the GD dataset.

4.6.1 Data splits

We conduct a larger scale experiment than Chapter 3 by combining their 20 "Train" and "Dev" treebanks (18 distinct languages) in Table 3.3 as *training data*, and hold out the remaining 15 languages for testing. We tune the hyperparameters on the training languages with 5-fold cross-validation. That is, for each fold, we train the system on 4 folds and evaluate on the remaining 1. The split information is in Table 4.2. Similar to Table 3.3, we exclude some treebanks for evaluation (see the caption of Table 4.2), which is also slightly different from the set up in the original work (Wang and Eisner, 2017), where no treebanks are excluded. When augmenting the 16 real languages with GD languages, we include only GD languages that are generated by "mixing-and-matching" those 16 languages, which means that we add $16 \times 17 \times 17 = 4624$ synthetic languages.[11]

[11] Why $16 \times 17 \times 17$? As Section 3.5 explains, each GD treebank is obtained from the UD treebank of some *substrate* language S by stochastically permuting the dependents of verbs and nouns to respect typical orders in the *superstrate* languages R_{V} and R_{N} respectively. There are 16 choices for S. There are 17 choices for R_{V} (respectively R_{N}), including $R_{\text{V}} = S$ ("self-permutation") and $R_{\text{V}} = \emptyset$ ("no permutation").

Split	Family	Sub-Family	Language	Treebank ID
Train1	Indo-European	Germanic	Danish	da
	Indo-European	Germanic	Norwegian	no
	Indo-European	Greek	Ancient Greek	grc_proiel
	Afro-Asiatic	Semitic	Arabic	ar
Train2	Indo-European	Greek	Ancient Greek	grc
	Indo-European	Romance	Portuguese	pt
	Indo-European	Slavic	Czech	cs
	Uralic	Finnic	Estonian	et
Train3	Indo-European	Germanic	German	de
	Indo-European	Germanic	Gothic	got
	Indo-European	Latin	Latin	la_proiel
	Indo-European	Romance	Italian	it
Train4	Indo-European	Latin	Latin	la_itt
	Indo-European	Romance	French	fr
	Indo-European	Slavic	Bulgarian	bg
	Uralic	Finnic	Finnish	fi
Train5	Indo-European	Germanic	Dutch	nl
	Indo-European	Germanic	English	en
	Indo-European	Indic	Hindi	hi
	Indo-European	Romance	Spanish	es
Test	Indo-European	Celtic	Irish	ga
	Indo-European	Germanic	Swedish	sv
	Indo-European	Greek	Greek	el
	Indo-European	Iranian	Persian	fa
	Indo-European	Latin	Latin	la
	Indo-European	Romance	Romanian	ro
	Indo-European	Slavic	Croatian	hr
	Indo-European	Slavic	Old Church Slavonic	cu
	Indo-European	Slavic	Polish	pl
	Indo-European	Slavic	Slovenian	sl
	Uralic	Finnic	Finnish	fi_ftb
	Uralic	Ugric	Hungarian	hu
	Afro-Asiatic	Semitic	Hebrew	he
	Austronesian	-	Indonesian	id
	Basque	-	Basque	eu
	Dravidian	Southern	Tamil	ta
	Japanese	-	Japanese	ja_ktc

Table 4.2: Data split of the 37 real treebanks. Different from Table 3.3, we group the treebanks by their split information. (Our "Train," on which we do 5-fold cross-validation, contains both their (Table 3.3) "Train" and "Dev" languages.) We follow the principle of Table 3.3 and does not test on the fi_ftb or Latin treebanks because other treebanks of those languages appeared in training data. Specifically, la_proiel and la_itt fall in "Train3" and "Train4", respectively. For the same reason, Table 4.8 does not show cross-validation development results on these Latin treebanks—nor on the grc_proiel and grc treebanks, which fall in "Train1" and "Train2", respectively. This results 16 training languages to be evaluated for cross-valuation. All the excluded treebanks are marked in red. In the final test, we will use the model trained on all *20* "Train" treebanks.

Each GD treebank $\boldsymbol{y}$ provides a standard split into train/dev/test portions. We primarily restrict ourselves to the train portions (saving the gold trees from the dev and test portions to tune and evaluate some future grammar induction system that consults our typological predictions). For example, we write $\boldsymbol{x}_{\text{train}}$ for the POS-tagged sentences in the "train" portion, and $\boldsymbol{\tau}(\boldsymbol{y}_{\text{train}})$ for the empirical probabilities derived from their gold trees.

We always train the model to predict $\boldsymbol{\tau}(\boldsymbol{y}_{\text{train}})$ from $\boldsymbol{x}_{\text{train}}$ on each *training language*. To evaluate on a *held-out language* during cross-validation, we can measure how well the model predicts $\boldsymbol{\tau}(\boldsymbol{y}_{\text{train}})$ given $\boldsymbol{x}_{\text{train}}$.[12] For our final test, we evaluate on the 15 test languages using a model trained on all training languages (20 treebanks for UD, plus $20 \times 21 \times 21 = 8840$ when adding GD) with the chosen hyperparameters. To evaluate on a *test language*, we again measure how well the model predicts $\boldsymbol{\tau}(\boldsymbol{y}_{\text{train}})$ from $\boldsymbol{x}_{\text{train}}$.

4.6.2 Comparison of architectures

Table 4.3 shows the cross-validation losses (Equation (4.1)) that are achieved by different scoring architectures. We compare the results when the model is trained on real languages (the "UD" column) versus on real languages plus synthetic languages (the "+GD" column).

The $\boldsymbol{\psi}_{\text{H}}$ models here use a subset of the hand-engineered features, selected by the experiments in Section 4.6.3 below and corresponding to Table 4.4 line 8.

[12]In actuality, we measured how well it predicts $\boldsymbol{\tau}(\boldsymbol{y}_{\text{dev}})$ given $\boldsymbol{x}_{\text{dev}}$. That was a slightly less sensible choice. It may have harmed our choice of hyperparameters, since dev is smaller than train and therefore $\boldsymbol{\tau}(\boldsymbol{y}_{\text{dev}})$ tends to have greater sampling error. Another concern is that our typology system, having been specifically tuned to predict $\boldsymbol{\tau}(\boldsymbol{y}_{\text{dev}})$, might provide an unrealistically accurate estimate of $\boldsymbol{\tau}(\boldsymbol{y}_{\text{dev}})$ to some future grammar induction system that is being cross-validated against the same dev set, harming that system's choice of hyperparameters as well.

Architecture		ε-insensitive loss	
Scoring	Depth	UD	+GD
Expected count (EC)	-	**0.113**	**0.109**
Hand-engineered features ($\boldsymbol{\psi}_{\mathrm{H}}$)	0	0.066	**0.041***
Hand-engineered features ($\boldsymbol{\psi}_{\mathrm{H}}$)	1	**0.058***	**0.041***
Hand-engineered features ($\boldsymbol{\psi}_{\mathrm{H}}$)	3	**0.067**	**0.054**
Neural features ($\boldsymbol{\psi}_{\mathrm{N}}$)	1	**0.069***	**0.047**
Combination	1	0.058*	**0.035***

Table 4.3: Average expected loss over 16 training languages, computed by 5-fold cross-validation. The first column indicates whether we score using hand-engineered features ($\boldsymbol{\psi}_{\mathrm{H}}$), neural features ($\boldsymbol{\psi}_{\mathrm{N}}$), or a combination (see end of Section 4.5.3). As a baseline, the first line evaluates the EC (expected count) heuristic from Section 4.4. Within each *column*, we star the best (smallest) result as well as all results that are not significantly worse. For each comparison between UD and +GD, we boldface the better (lower) result, or both if they are not significantly different. All the statistical significance tests are under paired permutation test over languages with $p < 0.05$.

Although Figure 4.1 and eq. (4.7) presented an "depth-1" scoring network with one hidden layer, Table 4.3 also evaluates "depth-d" architectures with d hidden layers. The depth-0 architecture simply predicts each directionality separately using logistic regression (although our training objective is not the usual convex log-likelihood objective).

Some architectures are better than others. We note that the hand-engineered features outperform the neural features—though not significantly, since they make complementary errors—and that combining them is best. However, the biggest benefit comes from augmenting the training data with GD languages; this consistently helps more than changing the architecture.

4.6.3 Contribution of different feature classes

To understand the contribution of different hand-engineered features, we performed forward selection tests on the depth-1 system, including only some of the features. In

ID	Features	Length	Loss (+GD)
0	∅	—	0.085
1	conditional	40	0.065
2	joint	40	0.057
3	PMI	40	**0.044**
4	asymmetry	40	0.046
5	rows 3+4	40	**0.043**
6	row 5+b	40	**0.042**
7	row 5+t	40	**0.043**
8*	row 5+b+t	40	**0.041**
9	row 8	10	0.048
10	row 8	10+40	**0.041**

Table 4.4: Cross-validation losses with different subsets of hand-engineered features from Section 4.5.3. "∅" is a baseline with no features (bias feature only), so it makes the same prediction for all languages. "conditional" = $\pi^w_{t|s}$ features, "joint" = $\pi^w_{t|s} \cdot \pi^w_s$ features, "PMI" = $\pi^w_{t|s} // \pi^w_t$ and $\pi^w_t // \pi^w_{t|s}$ features, "asymmetry" = $\pi^w_{t|s} // \pi^{-w}_{t|s}$ features, "b" are the features superscripted by b, and "t" are the features with truncated window. "+" means concatenation. The "Length" field refers to length thresholding (see Section 4.5.4). The system in the starred row is the one that we selected for row 2 of Table 4.3.

all cases, we trained in the "+GD" condition. The results are shown in Table 4.4. Any class of features is substantially better than baseline, but we observe that most of the total benefit can be obtained with just PMI or asymmetry features. Those features indicate, for example, whether a verb tends to attract nouns to its right or left. We did not see a gain from length thresholding.

4.6.4 Robustness to noisy input

We also tested our directionality prediction system on noisy input (without retraining it on noisy input). Specifically, we tested the depth-1 ψ_{H} system. This time, when evaluating on the dev split of a held-out language, we provided a noisy version of that input corpus that had been retagged by an automatic POS tagger (Nguyen et al., 2014), which was trained on just 100 gold-tagged sentences from the train split of that language. The average tagging accuracy over the 16 languages was only 77.91%. Nonetheless, the “UD”-trained and “+GD”-trained systems got respective losses of 0.06 and 0.046—nearly as good as in Table 4.3, which used gold POS tags.

4.6.5 Hyperparameter settings

For each result in Table 4.3–4.4, the hyperparameters were chosen by grid search on the cross-validation objective (and the table reports the best result). For the remaining experiments, we select the depth-1 combined models (Equation (4.11)) for both “UD” and “+GD,” as they are the best models according to Table 4.3.

The hyperparameters for the selected models are as follows: When training with “UD,” we took $\alpha = 1$ (which ignores $\boldsymbol{\psi}_{\mathrm{N}}$), with hidden layer size $h = 256$, $\sigma =$ sigmoid, L2_coeff $= 0$ (no L2 regularization), and dropout $= 0.2$. When training with “+GD,” we took $\alpha = 0.6$, with different hyperparameters for the two interpolated models: $\boldsymbol{\psi}_{\mathrm{H}}$ uses $h = 128$, $\sigma =$ sigmoid, L2_coeff $= 0$, and dropout $= 0.4$, while $\boldsymbol{\psi}_{\mathrm{N}}$ uses $h = 128$, emb_size $= 128$, rnn_size $= 32$, $\sigma =$ relu, L2_coeff $= 0$, and dropout $= 0.2$. For both “UD” and “+GD”, we use $\lambda = 1$ for the smoothing in Footnote 7.

4.6.6 Comparison with grammar induction

Grammar induction is an alternative way to predict word order typology. Given a corpus of a language, we can first use grammar induction to parse it into dependency trees, and then estimate the directionality of each dependency relation type based on these (approximate) trees.

However, what are the dependency relation types? Current grammar induction

	MS13	N10	EC	∅	UD	+GD
loss	0.156	0.134	0.110	0.093	0.090	**0.044**

Table 4.5: Cross-validation average expected loss of the two grammar induction methods, MS13 (Mareček and Straka, 2013) and N10 (Naseem et al., 2010), compared to the EC heuristic of Section 4.4 and our architecture of Section 4.5 (the version from the last line of Table 4.3). In these experiments, the dependency relation types are ordered POS pairs. N10 harnesses prior linguistic knowledge, but its improvement upon MS13 is not statistically significant. Both grammar induction systems are *significantly* worse than the rest of the systems, including even our two baseline systems, namely EC (the "expected count" heuristic from Section 4.4) and ∅ (the no-feature baseline system from Table 4.4 line 0). Like N10, these baselines make use of some cross-linguistic knowledge, which they extract in different ways from the training treebanks. Among our own 4 systems, EC is significantly worse than all others, and +GD is significantly better than all others. (Note: When training the *baselines*, we found that including the +GD languages—a bias-variance tradeoff— harmed EC but helped ∅. The table reports the better result in each case.)

systems produce unlabeled dependency edges. Rather than try to obtain a UD label like $r = \texttt{amod}$ for each edge, we label the edge deterministically with a POS pair such as $r = (parent = \texttt{NOUN}, child = \texttt{ADJ})$. Thus, we will attempt to predict the directionality of each POS-pair relation type. For comparison, we retrain our supervised system to do the same thing.

For the grammar induction system, we try the implementation of DMV with stop-probability estimation by Mareček and Straka (2013), which is a common baseline for grammar induction (Le and Zuidema, 2015) because it is language-independent, reasonably accurate, fast, and convenient to use. We also try the grammar induction system of Naseem et al. (2010), which is the state-of-the-art system on UD (Noji, Miyao, and Johnson, 2016). Naseem et al. (2010)'s method, like ours, has prior knowledge of what typical human languages look like.

Table 4.5 shows the results. Compared to Mareček and Straka (2013), Naseem et al. (2010) gets only a small (insignificant) improvement—whereas our "UD" system

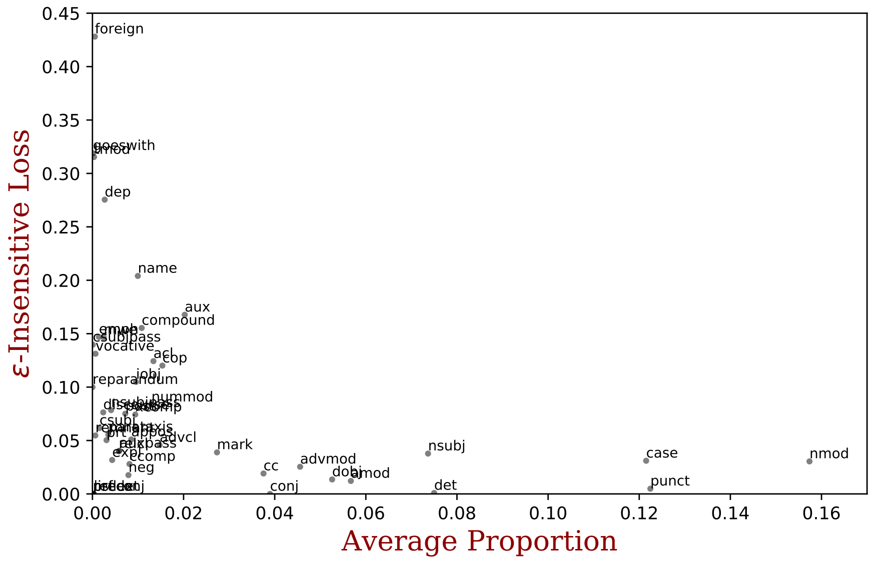


Figure 4.3: Cross-validation loss broken down by relation. We plot each relation r with x coordinate = the proportion of r in the average training corpus = $\text{mean}_{\ell \in \text{Train}}\, p_{\text{train}}(r \mid \ell) \in [0,1]$, and with y coordinate = the weighted average $\sum_{\ell \in \text{Heldout}} w_\ell(r)\, \text{loss}_\varepsilon(\hat{p}_{\text{dev}}(\rightarrow|r,\ell), p_{\text{dev}}(\rightarrow|r,\ell))$ (see Equation (4.13)).

halves the loss, and the "+GD" system halves it again. Even our baseline systems are significantly more accurate than the grammar induction systems, showing the effectiveness of casting the problem as supervised prediction.

4.6.7 Fine-grained analysis

Beyond reporting the aggregate cross-validation loss over the 16 training languages, we break down the cross-validation predictions by relation type. Figure 4.3 shows that the *frequent* relations are all quite predictable. Figure 4.4 shows that our success is not just because the task is easy—on relations whose directionality varies by language, so that a baseline method does poorly, our system usually does well.

To show that our system is behaving well across languages and not just on average, we zoom in on 5 relation types that are particularly common or of particular interest

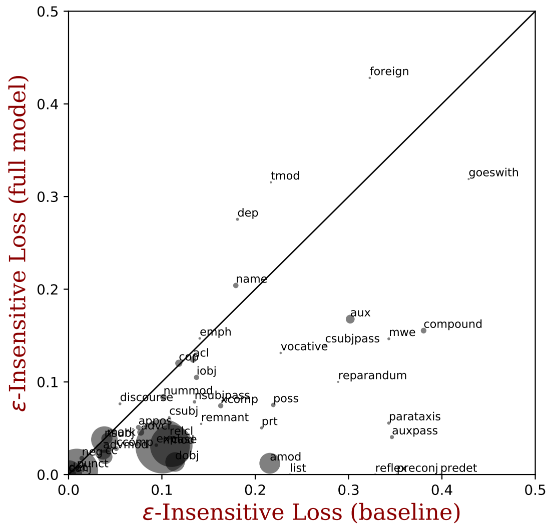


Figure 4.4: The y coordinate is the average loss of our model (Table 4.4 line 8), just as in Figure 4.3, whereas the x coordinate is the average loss of a simple baseline model $\varnothing$ that ignores the input corpus (Table 4.4 line 0). Relations whose directionality varies more by language have higher baseline loss. Relations that beat the baseline fall below the diagonal line. The marker size for each relation is proportional to the x-axis in Figure 4.3.

to linguistic typologists. These 5 relations together account for 47% of all relation tokens in the average language: `nmod` = noun-nominal modifier order, `nsubj` = subject-verb order (feature 82A in the World Atlas of Language Structures), `dobj` = object-verb order (83A), `amod` = adjective-noun order (87A), and `case` = placement of both adpositions and case markers (85A).

As shown in Figure 4.5, most points in the first five plots fall in or quite near the desired region. We are pleased to see that the predictions are robust when the training data is unbalanced. For example, the `case` relation points leftward in most real languages, yet our system can still predict the right directionality of hi, et and fi. The credit goes to the diversity of our training set, which contains various synthetic `case`-right languages: the system fails on these three languages if we train on real languages only. That said, apparently our training set is still not diverse enough to do

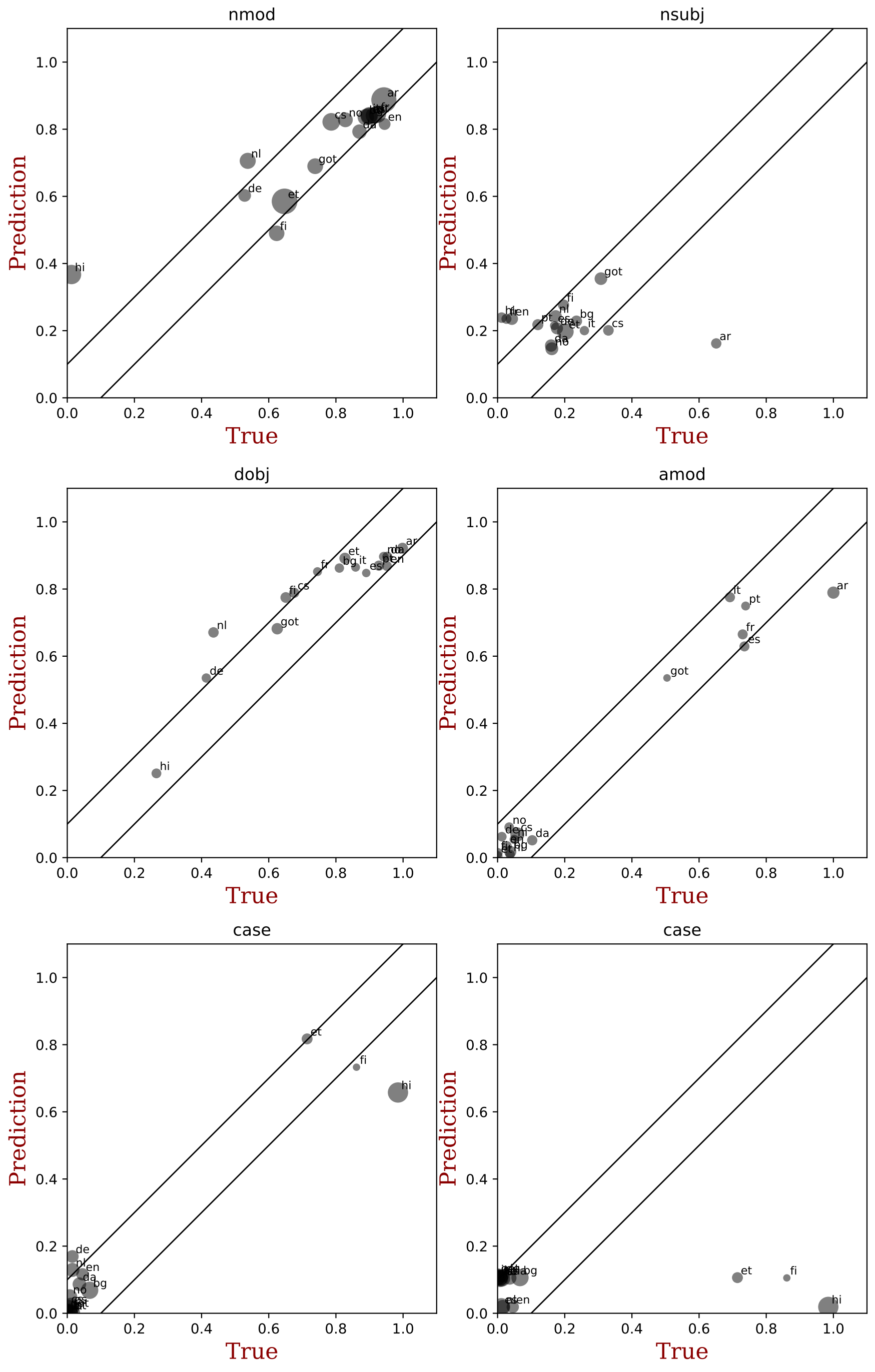


Figure 4.5: Scatterplots of predicted vs. true directionalities (by cross-validation). In the plot for relation type r, each language appears as a marker at $(p^*, \hat{p})$ (see Section 4.3), with the marker size proportional to $w_\ell(r)$ (see Equation (4.13)). Points that fall between the solid lines ($|\hat{p} - p^*| \leq \varepsilon$) are considered "correct," by the definition of ε-insensitive loss. The last plot (bottom right) shows worse predictions for `case` when the model is trained on UD only.

well on the outlier ar (Arabic); see Figure 3.4.

4.6.8 Binary classification accuracy

Besides ε-insensitive loss, we also measured how the systems perform on the coarser task of binary classification of relation direction. We say that relation r is dominantly "rightward" in language ℓ iff $p(\rightarrow\mid r,\ell) > 0.5$. We say that a system predicts "rightward" according to whether $\hat{p}(\rightarrow\mid r,\ell) > 0.5$.

We evaluate whether this binary prediction is correct for each of the 20 most frequent relations r, for each held-out language ℓ, using 5-fold cross-validation over the 16 training languages as in the previous experiment. Tables 4.6 and 4.7 respectively summarize these results by relation (equal average over languages) and by language (equal average over relations). Keep in mind that these systems had not been specifically trained to place relations on the correct side of the artificial 0.5 boundary.

Binary classification is an easier task. It is easy because, as the $\varnothing$ column in Table 4.6 indicates, most relations have a clear directionality preference shared by most of the UD languages. As a result, the better models with more features have less opportunity to help. Nonetheless, they do perform better, and the EC heuristic continues to perform worse.

In particular, EC fails significantly on `dobj` and `iobj`. This is because `nsubj`, `dobj`, and `iobj` often have different directionalities (e.g., in SVO languages), but the EC heuristic will tend to predict the same direction for all of them, according to whether `NOUN`s tend to precede nearby `VERB`s.

Relation	Rate	EC	∅	UD	+GD
nmod	0.16	0.94	0.94	0.94	0.94
punct	0.12	0.94	0.94	0.94	0.94
case	0.12	0.75	0.81	0.81	1.00
nsubj	0.08	0.94	0.94	0.94	0.94
det	0.07	0.81	0.94	0.94	0.94
amod	0.06	0.63	0.63	0.75	1.00
dobj	0.05	0.69	0.81	0.81	0.88
advmod	0.04	0.88	0.81	0.81	0.81
conj	0.04	1.00	1.00	1.00	1.00
cc	0.04	1.00	1.00	1.00	1.00
mark	0.03	0.94	0.94	0.94	0.94
aux	0.02	1.00	0.75	0.81	0.75
cop	0.02	0.81	0.81	0.75	0.81
advcl	0.01	0.94	0.94	0.94	0.94
nummod	0.01	0.94	0.94	0.94	0.94
acl	0.01	0.38	0.81	0.81	0.75
compound	0.01	0.44	0.44	0.44	0.38
xcomp	0.01	0.94	0.94	0.94	1.00
name	0.01	0.50	0.69	0.69	0.63
iobj	0.01	0.56	0.50	0.50	0.56
Average	-	0.8	0.83	0.83	0.86

Table 4.6: Accuracy on the simpler task of binary classification of relation directionality. The most common relations are shown first: the "Rate" column gives the average rate of the relation across the 16 training languages (like the x coordinate in Figure 4.3).

4.6.9 Final evaluation on test data

All previous experiments were conducted by cross-validation on the 16 treebanks. We now train the system on all 20 treebanks, and report results on the 15 blind test languages (Table 4.8). In our evaluation metric (Equation (4.1)), $\mathcal{R}$ includes all 57 relation types that appear in training data, plus a special UNK type for relations that appear only in test data. The results range from good to excellent, with synthetic data providing consistent and often large improvements.

These results could potentially be boosted in the future by using an even larger

Language	EC	∅	UD	+GD
Arabic	0.75	0.7	0.65	0.75
Bulgarian	0.75	0.9	0.9	0.9
Czech	0.85	0.95	0.95	0.9
Danish	0.8	0.9	0.9	0.9
German	0.9	0.85	0.85	0.9
English	0.85	0.95	0.95	0.95
Spanish	0.9	0.9	0.95	0.95
Estonian	0.7	0.7	0.7	0.7
Finnish	0.7	0.85	0.85	0.8
French	0.85	0.9	0.9	0.9
Gothic	0.7	0.75	0.8	0.8
Hindi	0.6	0.45	0.45	0.7
Italian	0.85	0.85	0.85	0.9
Dutch	0.9	0.8	0.8	0.85
Norwegian	0.9	0.95	0.95	0.9
Portuguese	0.8	0.85	0.9	0.9
Average	0.8	0.83	0.83	0.86

Table 4.7: Accuracy on the simpler task of binary classification of relation directionality for each training language. A detailed comparison shows that EC is *significantly* worse than +GD (paired permutation test over the 16 languages, $p < 0.05$). The difference among ∅, UD and +GD is *insignificant*, which suggests that this is an easier task where weak models might suffice.

and more diverse training set. In principle, when evaluating on any one of our real languages, one could train a system on *all* of the rest (plus the galactic languages derived from them), not just 20. Moreover, the Universal Dependencies collection has continued to grow beyond the 37 treebanks used here. Finally, our current setup extracts only one training example from each (real or synthetic) language. One could easily generate a variant of this example each time the language is visited during stochastic optimization, by bootstrap-resampling its training corpus (to add "natural" variation) or subsampling it (to train the predictor to work on smaller corpora). In the case of a synthetic language, one could also generate a corpus of new trees each time

Test			Train		
Language	UD	+GD	Language	UD	+GD
Basque	0.25	0.077	Arabic	0.116	0.056
Croatian	0.062	0.012	Danish	0.024	0.017
Greek	0.056	0.01	Norwegian	0.008	0.011
Hebrew	0.079	0.034	Czech	0.025	0.014
Hungarian	0.119	0.101	Estonian	0.055	0.015
Indonesian	0.099	0.073	Portuguese	0.038	0.004
Irish	0.181	0.154	German	0.046	0.027
Japanese	0.247	0.08	Gothic	0.008	0.03
Old Church Slavonic	0.024	0.029	Italian	0.011	0.01
Persian	0.22	0.121	Bulgarian	0.037	0.015
Polish	0.056	0.022	Finnish	0.069	0.07
Romanian	0.029	0.009	French	0.024	0.02
Slovenian	0.015	0.031	Dutch	0.069	0.064
Swedish	0.012	0.007	English	0.025	0.036
Tamil	0.238	0.052	Hindi	0.363	0.173
			Spanish	0.012	0.008
Test Average	0.112	0.054*	All Average	0.084	0.045*

Table 4.8: Our final comparison on the 15 test languages appears at left. We ask whether the average expected loss on these 15 real target languages is reduced by augmenting the training pool of 20 UD languages with +20*21*21 GD languages. For completeness, we extend the table with the cross-validation results on the training pool, which includes 16 training languages grouped by 5 folds (separated by dashed lines). The “Average” lines report the average of 15 test or 31 training+testing languages. We mark both “+GD” averages with “*” as they are significantly better than their “UD” counterparts (paired permutation test by language, $p < 0.05$).

the language is visited (by re-running the stochastic permutation procedure, instead of reusing the particular permutation released by the Galactic Dependencies project).

4.7 Related Work

Typological properties can usefully boost the performance of cross-linguistic systems (Bender, 2009; O’Horan et al., 2016). These systems mainly aim to annotate low-resource languages with help from models trained on similar high-resource languages.

Naseem, Barzilay, and Globerson (2012) introduce a "selective sharing" technique for generative parsing, in which a Subject-Verb language will use parameters shared with other Subject-Verb languages. Täckström, McDonald, and Nivre (2013) and Zhang and Barzilay (2015) extend this idea to discriminative parsing and gain further improvements by conjoining regular parsing features with typological features. The cross-linguistic neural parser of Ammar et al. (2016) conditions on typological features by supplying a "language embedding" as input. Zhang et al. (2012) use typological properties to convert language-specific POS tags to UD POS tags, based on their ordering in a corpus.

Moving from engineering to science, linguists seek *typological universals* of human language (Greenberg, 1963; Croft, 2002; Song, 2014; Hawkins, 2014), e.g., "languages with dominant Verb-Subject-Object order are always prepositional." Dryer and Haspelmath (2013) characterize 2679 world languages with 192 typological properties. Their WALS database can supply features to NLP systems (see previous paragraph) or gold standard labels for typological classifiers. Daumé III and Campbell (2007) take WALS as input and propose a Bayesian approach to discover new universals. Georgi, Xia, and Lewis (2010) impute missing properties of a language, not by using universals, but by backing off to the language's typological cluster. Murawaki (2015) use WALS to help recover the evolutionary tree of human languages; Daumé III (2009) considers the geographic distribution of WALS properties.

Attempts at automatic typological classification are relatively recent. Lewis and Xia (2008) predict typological properties from induced trees, but guess those trees from aligned bitexts, not by monolingual grammar induction as in Section 4.6.6. Liu (2010) and Futrell, Mahowald, and Gibson (2015) show that the directionality of (gold) dependencies is indicative of "basic" word order and freeness of word order. Those papers predict typological properties from trees that are automatically (noisily) annotated or manually (expensively) annotated. In addition, parallel data is shown to be useful Östling (2015) and Malaviya, Neubig, and Littell (2017), where most work used the machine translation techniques to infer alignment information for analyzing for predicting word order typology.

An alternative is to predict the typology directly from raw or POS-tagged text, as we do. Saha Roy et al. (2014) first explored this idea, building a system that correctly predicts adposition typology on 19/23 languages with only word co-occurrence statistics. Zhang et al. (2016) evaluate semi-supervised POS tagging by asking whether the induced tag sequences can predict typological properties. Their prediction approach is supervised like ours, although developed separately and trained on different data.

They more simply predict 6 binary-valued WALS properties, using 6 independent binary classifiers based on POS bigram and trigrams. Our task is rather close to grammar induction, which likewise predicts a set of real numbers giving the relative probabilities of competing syntactic configurations. Most previous work on grammar induction begins with maximum likelihood estimation of some generative model—such as a PCFG (Lari and Young, 1990; Carroll and Charniak, 1992) or dependency grammar (Klein and Manning, 2004)—though it may add linguistically-informed inductive bias (Ganchev et al., 2010; Naseem et al., 2010). Most such methods use local search and must wrestle with local optima (Spitkovsky, Alshawi, and Jurafsky, 2013). Fine-grained typological classification might supplement this approach, by cutting through the initial combinatorial challenge of establishing the basic word order properties of the language. In this work we only quantify the directionality of each relation *type*, ignoring how *tokens* of these relations interact locally to give coherent parse trees. Grammar induction methods like EM could naturally consider those local interactions for a more refined analysis, when guided by our predicted global directionalities.

4.8 Conclusions and Future Work

In this chapter, we introduced a typological classification task as a testbed of our AB inference function, which attempts to extract quantitative knowledge about a language's syntactic structure from its surface forms (POS tag sequences). As far as we know, we are the first to utilize synthetic languages to train a learner for real languages: this move yielded substantial benefits.[13]

Figure 4.5 shows that we rank held-out languages rather accurately along a spectrum of directionality, for several common dependency relations. Table 4.8 shows that if we jointly predict the directionalities of *all* the relations in a new language, most of those numbers will be quite close to the truth (low aggregate error, weighted by relation frequency). This holds promise for aiding grammar induction.

Our trained model is robust when applied to noisy POS tag sequences. In the future, however, we would like to make similar predictions from raw word sequences. That will require features that abstract away from the language-specific vocabulary. Although recurrent neural networks in this chapter did not show a clear advantage over hand-engineered features, they might be useful when used with word embeddings.

Finally, the output of our system has downstream uses. Several NLP tasks have benefited from typological features (Section 4.1). By using end-to-end training, our

[13]See Chapter 3 for a review of using synthetic training data elsewhere in machine learning.

methods could be tuned to extract typological features that are particularly useful for some task.

Chapter 5

Unsupervised Dependency Parsing

Chapter 4 has shown our amortized Bayes (AB) inference function (Section 2.4.4) is effective at predicting the fine-grained syntactic typology of a language. However, that task only finds global typological information: it did not establish *which* 70% of the direct objects fell to the right of their verbs, let alone identify which nouns were in fact direct objects of which verbs. That requires a token-level analysis of each sentence, which is unsupervised parsing—the final goal of this thesis as we will undertake in this chapter. In other words , instead of predicting typological properties of a language as Chapter 4 did, we will predict the actual treebank $\hat{\boldsymbol{y}}$ corresponding to the observed corpus $\boldsymbol{x}$. The experimental results will show that our best method improved attachment scores on held-out test languages by an average of 5.6 percentage points over past work that does not inspect the unparsed data (McDonald, Petrov, and Hall, 2011), and by 20.7 points over past "grammar induction" work that does not use training languages (Naseem et al., 2010).

5.1 Task Formulation

The positive results of Chapter 4 demonstrate that there are indeed surface clues to some linguistic properties in the input corpus, at least if it is POS-tagged. In this chapter, we are interested in studying whether the input corpus has useful information to form a language-specific parser, which is arguably a more challenging task than typology prediction.

An unsupervised parser for language ℓ is built without any gold parse trees for ℓ. Starting from Algorithm 2, like Chapter 4, we assume a corpus $\boldsymbol{x}$ of unparsed but POS-tagged sentences of ℓ is available for input.

One difference is that Chapter 4 predicts $\boldsymbol{\tau}(\boldsymbol{y})$, which is a fine-grained typology vector derived from parse trees. In contrast, this chapter directly predicts the parse trees $\boldsymbol{y}$ for the input sentences $\boldsymbol{x}$. As Equation (2.11) explains, another difference is that we assume additional "side information" $\boldsymbol{u}$ for extracting surface cues about the language that are useful to the parser.

Overall, our approach is to train a "language-agnostic" parser—one that does not know what language ℓ it is parsing in. Taking $(\boldsymbol{x}, \boldsymbol{u})$ as input, it produces parse trees $\hat{\boldsymbol{y}} = \mathrm{e}_{\boldsymbol{\Theta}}(\boldsymbol{x}, \boldsymbol{u})$. The parameters $\boldsymbol{\Theta}$ are shared by all languages. To learn them, we will allow ℓ to range over training languages, and then test our ability to parse when ℓ ranges over novel test languages.

Following Algorithm 3, we train $\boldsymbol{\Theta}$ on a collection of $(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{y})$ tuples , each of which corresponds to a UD or GD training language ℓ. In other words, as discussed in Section 1.4, our system is trained to match how linguists annotate the training languages. We can therefore directly define the per-language loss (Line 5 in Algorithm 3) for each training language ℓ as

$$\text{Loss}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \underset{(x,y)\in(\boldsymbol{x},\boldsymbol{y})}{\text{mean}} \text{loss}(\underbrace{\text{Parse}_{\boldsymbol{\Theta}}(x; \boldsymbol{u})}_{\hat{y}}, y) \tag{5.1}$$

where loss$(\ldots)$ is a task-specific per-sentence loss (defined in Section 5.5.1) that evaluates the parser's output $\hat{y} \overset{\text{def}}{=} \text{Parse}_{\boldsymbol{\Theta}}(x; \boldsymbol{u})$ on sentence x against x's correct tree y.

Our parser $\text{Parse}_{\boldsymbol{\Theta}}(x; \boldsymbol{u})$ has two stages. First, it uses a neural network to extract statistics $\text{T}(\boldsymbol{u})$ from $\boldsymbol{u}$ that are informative about the syntactic structure of ℓ, to guide us in parsing POS-tagged sentences of ℓ. $\text{T}(\boldsymbol{u}) \in \mathbb{R}^m$ is a vector that represents the typological properties of ℓ and resembles the *language embedding* of Ammar et al. (2016). Then it parses sentence x while taking $\text{T}(\boldsymbol{u})$ as an additional input. We will give details of these two components in Sections 5.3 and 5.4. We assume that the input sentence x is given as a POS sequence: that is, our parser is *delexicalized*. This spares us from also needing language-specific lexical parameters associated with the specific vocabulary of each language, a problem that we leave to future work.

5.2 Related Work

5.2.1 Per-language learning

Many papers rely on some *universal learning procedure* to determine $\text{T}(\boldsymbol{u})$ (see Section 5.1) for a target language. For example, $\text{T}(\cdot)$ may be the Expectation-Maximization (EM) algorithm, yielding a PCFG $\text{T}(\boldsymbol{u})$ that fully determines a CKY parser (Carroll and Charniak, 1992; Klein and Manning, 2004). Since EM and CKY are fixed algorithms, this approach has no trainable parameters.

Grammar induction tries to turn an unsupervised corpus into a generative grammar. The approach of the previous paragraph is often modified to reduce model error or search error (Section 1.3). To reduce model error, many papers have used dependency grammar, with training objectives that incorporate notions like lexical attraction (Yuret, 1998) and grammatical bigrams (Paskin, 2001; Paskin, 2002). The dependency model with valence (DMV) (Klein and Manning, 2004) was the first method to beat a simple right-branching heuristic. Headden III, Johnson, and McClosky (2009), Spitkovsky, Alshawi, and Jurafsky (2012) and Blunsom and Cohn (2010) made the DMV more expressive by including more linguistic phenomenon (such as higher-order valency or punctuation). To reduce search error, strategies for eliminating or escaping local optima have included convexified objectives (Wang, Schuurmans, and Lin, 2008; Gimpel and Smith, 2012), smart initialization (Klein and Manning, 2004; Mareček and Straka, 2013), search bias (Smith and Eisner, 2005; Smith and Eisner, 2006; Naseem et al., 2010; Gillenwater et al., 2010), branch-and-bound search (Gormley and Eisner, 2013), and switching objectives (Spitkovsky, Alshawi, and Jurafsky, 2013).

Unsupervised parsing (which is also our task) tries to turn the same corpus directly into a treebank, without necessarily finding a grammar. We discuss some recent milestones here. Grave and Elhadad (2015) propose a transductive learning objective for unsupervised parsing, and a convex relaxation of it. (Jiang, Han, and Tu (2017) combined that work with grammar induction.) Martínez Alonso et al. (2017) create an unsupervised dependency parser that is formally similar to ours in that it uses cross-linguistic knowledge as well as statistics computed from a corpus of POS sequences in the target language. However, its cross-linguistic knowledge is hand-coded: namely, the set of POS-to-POS dependencies that are allowed by the UD annotation scheme, and the typical directions for some of these dependencies. The only corpus statistic extracted from $\boldsymbol{u}$ is whether `ADP-NOMINAL` or `NOMINAL-ADP` bigrams are more frequent,[1] which distinguishes prepositional from postpositional languages. The actual parser starts by identifying the head word as the most "central" word according to a PageRank (Page et al., 1999) analysis of the graph of candidate edges, and proceeds by greedily attaching words of decreasing PageRank at lower depths in the tree.

5.2.2 Multi-language learning

This approach parses a "target" language using the treebanks of other resource-rich languages as "source" languages, which attacts recent attention (Zeman et al., 2017; Zeman et al., 2018) from the NLP community. There are two main variants.

Memory-based. This method trains a supervised parsing model on each source treebank. It uses these (delexicalized) source-language models to help parse the target sentence, favoring sources that are similar to the target language. A common similarity measure (Rosa and Žabokrtský, 2015a) considers the probability of the target language's POS-corpus $\boldsymbol{u}$ under a trigram language model of source-language POS sequences. Typological similarily has also been used for measuring (Shi et al., 2017), which assumes the typological information of the target language to be available.

Single-source transfer (SST) (Rosa and Žabokrtský, 2015a) simply uses the parser

[1] In our notation of Section 5.3.1, below, this asks whether $\sum_{t\in\{\texttt{NOUN},\texttt{PRON},\texttt{PROPN}\}} \pi^{w}_{t|\texttt{ADP}}$ is greater for $w = 1$ or $w = -1$.

for the *most similar* source treebank. Multi-source transfer (MST) (Rosa and Žabokrtský, 2015a) parses the target POS sequence with *each* of the source parsers, and then combines these parses into a consensus tree using the Chu-Liu-Edmonds algorithm (Chu, 1965; Edmonds, 1967). As a faster variant, model interpolation (Rosa and Žabokrtský, 2015b) builds a consensus *model* for the target language (via a weighted average of source models' parameters), rather than a consensus *parse* for each target sentence separately.

Memory-based methods require storing models for all source treebanks, which is expensive when we include thousands of GD treebanks (Chapter 3).

Model-based. This method trains a single *language-agnostic model*. McDonald, Petrov, and Hall (2011) train a delexicalized parser on the concatenation of all source treebanks, achieving a large gain over grammar induction. This parser can learn universals such as the preference for determiners to attach to nouns (which was hard-coded by Naseem et al. (2010)). However, when a parsing architecture is expressive enough, it is expected to parse a sentence x without being told the language ℓ or even a corpus $\boldsymbol{u}$, possibly by guessing properties of the language from the configurations it encounters in the single sentence x alone (Fisch, Guo, and Barzilay, 2019).

Further gains were achieved (Naseem, Barzilay, and Globerson, 2012; Täckström, McDonald, and Nivre, 2013; Zhang and Barzilay, 2015; Ammar et al., 2016) by *providing* the parser with about 10 typological properties of x's language—for example, whether direct objects generally fall to the right of the verb—as listed in the World Atlas of Linguistic Structures (Dryer and Haspelmath, 2013).

However, relying on WALS raises some issues. (1) The unknown language might not be in WALS.[2] (2) Some typological features are missing for some languages. (3) All the WALS features are categorical values, which loses useful information about tendencies (for example, how often the canonical word order is violated). (4) Not all WALS features are useful—only 56 of them pertain to word order, and only 8 of those have been used in past work. (5) With a richer parser (a stack LSTM dependency parser), WALS features do not appear to help at all on unknown languages (Ammar et al., 2016, Footnote 30).

In addition to pure memory-based or model-based methods, Smith et al. (2018) proposed a hybrid approach by parsing a target language with the parser trained on the concatenation of the source treebanks that are closely related to the target.

5.2.3 Exploiting parallel data

Some other work on generalizing from source to target languages assumes the availability of source-target parallel data, or bitext. Two uses:

Induction of multilingual word embeddings. Similar to universal POS tags, multilingual word embeddings serve as a universal representation that bridges the lexical differences among languages. Guo et al. (2016) proposed two approaches: (1) Training a variant of the skip-gram model (Mikolov et al., 2013) by using bilingual sets of context words. (2) Generating the embedding of each target word by averaging the embeddings of the source words to which it is aligned.

Annotation projection. Given aligned bitext, one can generate an approximate parse for a target sentence by "projecting" the parse tree of the corresponding source sentence. A target-language parser can then be trained from these approximate parses.

[2] 2,679 out of about 7,000 world languages are in WALS.

The idea was originally proposed by Yarowsky, Ngai, and Wicentowski (2001), and then applied to dependency parsing on low-resource languages Hwa et al., 2005; Ganchev, Gillenwater, and Taskar, 2009; Smith and Eisner, 2009; Tiedemann, 2014, *inter alia*. McDonald, Petrov, and Hall (2011) extend this approach to multiple source languages by *projected transfer*. Later work in this vein mainly tries to improve the approximate parses, including translating the source treebanks into the target language with an off-the-shelf machine translation system (Tiedemann, Agić, and Nivre, 2014; Tiedemann and Agić, 2016; Rosa and Mareček, 2018), augmenting the trees with weights (Agić et al., 2016), and using only partial trees with high-confidence alignments (Rasooli and Collins, 2015; Rasooli and Collins, 2017; Lacroix et al., 2016).

5.2.4 Situating our work

Our own approach can be categorized as model-based multi-language learning with no parallel text or target-side supervision. However, we also analyze an unparsed corpus $\boldsymbol{u}$ of the target language, as the per-language systems of Section 5.2.1 do. Our analysis of $\boldsymbol{u}$ does not produce a specialized target grammar or parser, but only extracts a target vector $\mathrm{T}(\boldsymbol{u})$ to be fed to the language-agnostic parser. The analyzer is trained jointly with the parser, over many languages. Recently, a similar idea has been proposed by Platanios et al. (2018) for *zero-shot* machine translation. Given a language pair, they feed the embeddings of its source and target languages into a *language-pair-agnostic* parameter generator (like our AB inference function) to *predict* a translation system that is adapted to this language pair.

5.3 The Typology Component

Chapter 4 extract typological properties of a language from its POS-tagged corpus $\boldsymbol{u}$, in effect predicting syntactic structure from superficial features. Similarly, we compute a hidden layer $\mathrm{T}(\boldsymbol{u})$ using a standard multilayer perceptron architecture, for example,

$$\mathrm{T}(\boldsymbol{u}) = \sigma(W\boldsymbol{\pi}(\boldsymbol{u}) + \mathbf{b}_W) \in \mathbb{R}^h \tag{5.2}$$

where $\boldsymbol{\pi}(\boldsymbol{u}) \in \mathbb{R}^d$ is the surface features of $\boldsymbol{u}$, $W \in \mathbb{R}^{h \times d}$ maps $\boldsymbol{\pi}(\boldsymbol{u})$ into a h-dimensional space, $\mathbf{b}_W \in \mathbb{R}^h$ is a bias vector, and σ is an element-wise activation function. While Equation (5.2) has only 1 layer, we explore versions with from 0 to 3 layers (where $\mathrm{T}(\boldsymbol{u}) = \boldsymbol{\pi}(\boldsymbol{u})$ in the 0-layer case). A 2-layer version is shown in Figure 5.1. The number of layers is chosen by cross-validation, as are h and the σ function.

5.3.1 Design of the surface features $\boldsymbol{\pi}(\boldsymbol{u})$

To define $\boldsymbol{\pi}(\boldsymbol{u})$, we used development data to select the following fast but effective subset of the features proposed in Section 4.5.3.

Hand-engineered features. Using the same notation as Section 4.5.3, the final hand-engineered $\boldsymbol{\pi}(\boldsymbol{u})$ includes:

- π_t^w, for each tag type t and each $w \in \{1, 3, 8, 100\}$. This quantity measures how frequently t appears in $\boldsymbol{u}$.
- $\pi_{t|s}^w // \pi_t^w$ and $\pi_{t|s}^{-w} // \pi_t^{-w}$, for each tag type pair s, t and each $w \in \{1, 3, 8, 100\}$.

Neural features. We use the same architecture as Figure 4.2, except the final neural

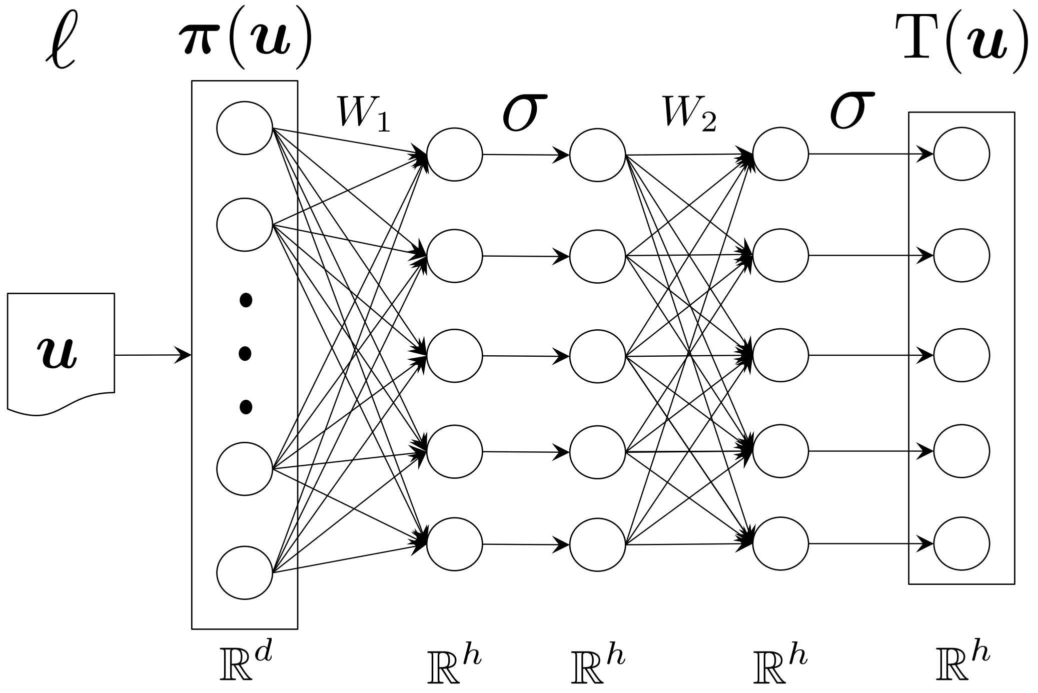


Figure 5.1: A 2-layer typology component. The bias vectors ($\mathbf{b}_W$) are suppressed for readability.

$\pi(\boldsymbol{u})$ only uses the average encoding of all sentences (average-pooling): that is, the average of all sentence-level configurations. We specifically use a gated recurrent unit (GRU) network (Cho et al., 2014b). The GRU is jointly trained with all other parameters in the system so that it focuses on detecting word order properties of $\boldsymbol{u}$ that are useful for parsing.

5.4 The Parsing Architecture

To construct $\mathrm{Parse}(x;\boldsymbol{u})$, we can extend any statistical parsing architecture $\mathrm{Parse}(x)$ to be sensitive to $\mathrm{T}(\boldsymbol{u})$. For our experiments, we extend the delexicalized graph-based implementation of the BIST parser (Kiperwasser and Goldberg, 2016)—an arc-factored dependency model with neural context features extracted by a bidirectional

LSTM. This recent parser was the state of the art when it was published.

Given a POS-sentence x and a corpus $\boldsymbol{u}$, our parser first computes an unlabeled projective tree

$$\operatorname*{argmax}_{y \in \mathcal{Y}(x)} \operatorname{score}(x, y; \boldsymbol{u}) \tag{5.3}$$

where, letting a range over the arcs in tree y,

$$\operatorname{score}(x, y; \boldsymbol{u}) = \sum_{a \in y} s(\phi(a; x, \boldsymbol{u})) \tag{5.4}$$

With this definition, the argmax in Equation (5.3) is computed efficiently by the algorithm of Eisner (1996).

$s(\cdot)$ is a neural scoring function on vectors,

$$s(\phi(\cdots)) = \mathbf{v} \tanh(V\phi(\cdots) + \mathbf{b}_V) \tag{5.5}$$

where V is a matrix, $\mathbf{b}_V$ is a bias vector, and $\mathbf{v}$ is a vector, all being parameters in $\boldsymbol{\Theta}$.

The function $\phi(a; x, \boldsymbol{u})$ extracts the feature vector of arc a given x and $\boldsymbol{u}$. BIST scores unlabeled arcs, so a denotes a pair (i, j)—the indices of the parent and child, respectively. We define

$$\phi(a; x, \boldsymbol{u}) = [\mathrm{B}(x, i; \mathrm{T}(\boldsymbol{u})); \mathrm{B}(x, j; \mathrm{T}(\boldsymbol{u}))] \tag{5.6}$$

which concatenates contextual representations of tokens i and j. $\mathrm{B}(x, i)$ is itself a concatenation of the hidden states of a left-to-right LSTM and a right-to-left LSTM (Graves, 2012) when each has read sentence x up through word i (really POS tag i). These LSTM parameters are included in $\boldsymbol{\Theta}$.

The POS tags in x are provided to the LSTMs as one-hot vectors. Crucially, $\mathrm{T}(\boldsymbol{u})$

is also provided to the LSTM at each step, as shown in Figure 5.2.

After selecting the best tree via Equation (5.3), we use each arc's ϕ vector again to predict its label. This yields the labeled tree $\hat{y} = \text{Parse}_{\mathbf{\Theta}}(x; \boldsymbol{u})$.

The only extension that this makes to BIST is to supply $\text{T}(\boldsymbol{u})$ to the BiLSTM.[3] This extension is not a significant slowdown at test time, since $\text{T}(\boldsymbol{u})$ only needs to be computed once per test language, not once per test sentence. Since $\text{T}(\boldsymbol{u})$ can be computed for any novel language at test time, this differs from the "many languages, one parser" architecture (Ammar et al., 2016), in which a test-time language must have been seen at training time or at least must have known WALS features.

Product of experts. Similar to Equation (4.11), we also consider a combined variant of the function Equation (5.5) for scoring arc a, namely

$$\alpha s_{\text{H}}(a) + (1 - \alpha) s_{\text{N}}(a) \tag{5.7}$$

where $s_{\text{H}}(a)$ and $s_{\text{N}}(a)$ are the scores produced by separately trained systems using, respectively, the hand-engineered and neural features from Section 5.3.1. Hyperparameter $\alpha \in [0, 1]$ is tuned on through cross-validation.

5.5 Training the System

5.5.1 Training objective

We exactly follow the training method of Kiperwasser and Goldberg (2016), who minimize a structured max-margin hinge loss (Taskar et al., 2004; McDonald, Crammer,

[3]An alternative would be to concatenate $\text{T}(\boldsymbol{u})$ with the representation computed by the BiLSTM. This gets empirically worse results, probably because the BiLSTM does not have advance knowledge of language-specific word order as it reads the sentence. We also tried an architecture that does both, with no notable improvement.

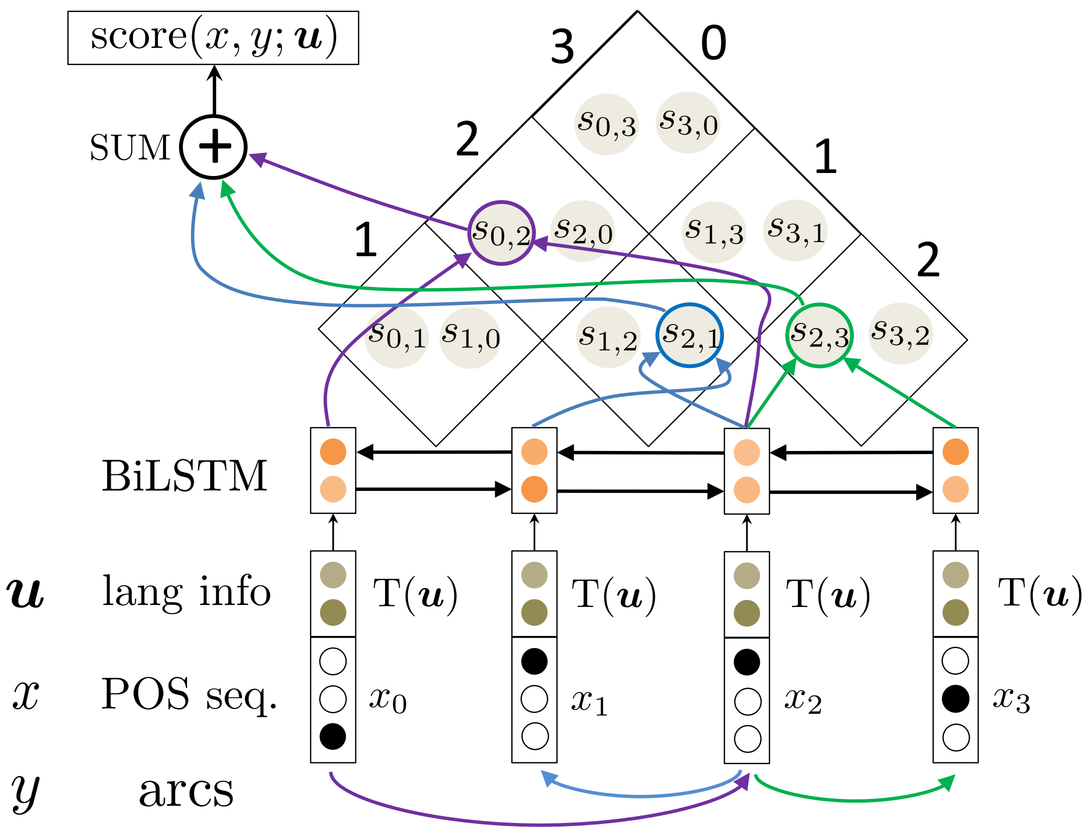

score$(x, y; \boldsymbol{u})$
SUM
3
0
2
1
1
2
$s_{0,3}$ $s_{3,0}$
$s_{0,2}$ $s_{2,0}$ $s_{1,3}$ $s_{3,1}$
$s_{0,1}$ $s_{1,0}$ $s_{1,2}$ $s_{2,1}$ $s_{2,3}$ $s_{3,2}$
BiLSTM
$\boldsymbol{u}$ lang info
T$(\boldsymbol{u})$
x POS seq.
x_0 x_1 x_2 x_3
y arcs

Figure 5.2: The architecture of the delexicalized graph-based BIST parser with the introduction of $\mathrm{T}(\boldsymbol{u})$, where $s_{i,j}$ in each cell is the arc score $s(\phi(a; x, \mathrm{T}(\boldsymbol{u}))$ from Equation (5.5). The root of the tree is always position 0, where x_0 is a distinguished "root" symbol that is prepended to the input sentence.

and Pereira, 2005; LeCun et al., 2007). We want the correct tree y to beat each tree y' by a margin equal to the number of errors in y' (we count spurious edges). Formally, loss$(x, y; \boldsymbol{u})$ is given by

$$\max(0, -\,\mathrm{score}(x, y; \boldsymbol{u}) +$$
$$\max_{y'} \Big(\underbrace{\mathrm{score}(x, y'; \boldsymbol{u})}_{\text{model score}} + \underbrace{\sum_{a \in y'} \mathbb{1}_{a \notin y}}_{\text{precision error}} \Big)) \qquad (5.8)$$

where a ranges over the arcs of a tree y, and $\mathbb{1}_{a \notin y}$ is an indicator that is 1 if $a \notin y$.

Thus, this loss function is high if there exists a tree y' that has a high score relative to y yet low precision.[4]

The training algorithm makes use of loss-augmented inference (Taskar et al., 2005), a variant on the ordinary inference of Equation (5.3). The most violating tree y' (in the $\max_{y'}$ above) is computed again by an arc-factored dependency algorithm (Eisner, 1996), where the score of any candidate arc a is $s(\phi(a; x, \boldsymbol{u})) + \mathbb{1}_{a \notin y}$.

Actually, the above method would only train the score function to predict the correct *unlabeled* tree as above (since a ranges over unlabeled arcs as before). In practice, we also jointly train the labeler to predict the correct labels on the gold arcs, using a separate hinge-loss objective. Because these two components share parameters through $\phi(a; x, \boldsymbol{u})$, this is a multi-task learning problem.

5.5.2 Training algorithm

Synthetic training data. Under our framework, each training example in Algorithm 3 is an entire language. We will experiment on augmenting our training dataset $\mathcal{L}_{\textbf{train}}$ with thousands of synthetic languages from the GD dataset (Chapter 3), as already discussed in Section 1.4.1.

Treating each language as a single large example during training would lead to slow SGD steps. Instead, we take our SGD examples to be individual sentences, by regarding Line 5 in Algorithm 3, and Equation (5.1) together as an objective averaged over sentences. Each example $(x, y, \boldsymbol{u})$ is sampled hierarchically, by first drawing a language ℓ from $\mathcal{L}_{\textbf{train}}$ and setting $\boldsymbol{u} = \boldsymbol{u}^{(\ell)}$, then drawing the sentence (x, y) uniformly from $(\mathbf{x}^{(\ell)}, \boldsymbol{y}^{(\ell)})$. We train using mini-batches of 100 sentences; each

[4]Formally, for this loss function to be used in equation Equation (5.1), we must interpret Parse_{Θ} in that equation as returning a forest of scored parses, not just a single parse.

mini-batch can mix many languages.

Encourage real languages. To sample ℓ from $\mathcal{L}_{\textbf{train}}$, we first flip a coin with weight $\beta \in [0, 1]$ to choose "real" vs. "synthetic," and then sample uniformly within that set. Why? The test sentences will come from real languages, so the synthetic languages are out-of-domain. Including them reduces variance but increases bias. We raise β to keep them from overwhelming the real languages.

Sample efficiently. The sentences (x, y) are stored in different files by language. To reduce disk accesses, we do not visit a file on each sample. Rather, for each language ℓ, we maintain in memory a *subset* of $(\mathbf{x}^{(\ell)}, \boldsymbol{y}^{(\ell)})$, obtained by reservoir sampling. Samples from $(\mathbf{x}^{(\ell)}, \boldsymbol{y}^{(\ell)})$ are drawn sequentially from this "chunk," and when it is used up we fetch a new chunk. We also maintain $\boldsymbol{u}^{(\ell)}$ and the hand-engineered features from $\boldsymbol{\pi}(\boldsymbol{u}^{(\ell)})$ in memory.

5.6 Experiments

5.6.1 Basic setup

Our data split, hyperparameter tuning, and evaluation follow the same setup as Chapter 4 (see Table 4.2). The UD and GD corpora provide a train/dev/test split of each treebank, denoted as $(\boldsymbol{x}_{\text{train}}, \boldsymbol{y}_{\text{train}})$, $(\boldsymbol{x}_{\text{dev}}, \boldsymbol{y}_{\text{dev}})$ and $(\boldsymbol{x}_{\text{test}}, \boldsymbol{y}_{\text{test}})$. Throughout this chapter, for both training and testing languages, we take $(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}_{\text{train}}, \boldsymbol{y}_{\text{train}})$. We take $\boldsymbol{u}$ to consist of all $\boldsymbol{x}_{\text{train}}$ sentences with ≤ 40 tokens.

5.6.2 Comparison among architectures

Table 5.1 shows the cross-validation parsing results over different systems discussed so far. For each architecture, we show the best average unlabeled attachment score (the

		UAS		LAS	
	System	UD	+GD	UD	+GD
	Single-source transfer (SST)	**66.22***	**65.70**	**50.40**	**50.54**
	Baseline	63.95	**67.97**	48.46	**52.78**
	Hand-engineered features (H)	64.83	**69.41**	49.41	**53.63**
	Neural features (N)	65.30	**70.06**	49.43	**54.19**
	Concatenation of H and N (H;N)	65.26	**69.62**	49.67	**53.68**
	Product-of-experts (H+N)	67.34*	**70.65***	52.02*	**55.18***
oracle features	Directionalities (T_D)	65.94	**70.01***	49.77	**53.43**
oracle features	WALS typology (T_W)	64.84	**69.75**	49.30	**53.79**

Table 5.1: Average parsing results over 16 languages, computed by 5-fold cross-validation. We compare training on real languages only (the UD column) versus augmenting with synthetic languages at $\beta = 0.2$ (the +GD column). Baseline is the ablated system that omits $T(\boldsymbol{u})$ (Section 5.6.2). SST is the single-source transfer approach (Section 5.2.2). H and N use only hand-engineered features or neural features, while H;N defines $\boldsymbol{\pi}(\boldsymbol{u})$ to concatenate both (Section 5.3.1) and H+N is the product-of-experts model (Section 5.4). T_D and T_W that incorporate oracle knowledge of the target-language syntax (Section 5.6.4). For each comparison between UD and +GD, we boldface the better (higher) result, or both if they are not significantly different (paired permutation test over languages with $p < 0.05$). In each column, we star the best result as well as all results that are not significantly worse.

UAS column) chosen by cross-validation, and the corresponding labeled attachment score (the LAS column). In brief, the main sources of improvement are twofold:

Synthetic languages. We observe that +GD consistently outperforms UD across all architectures. It even helps with the baseline system that we tried, which simply *ignores the target corpus* $\boldsymbol{u}^{(\ell)}$. In that system (similar to McDonald, Petrov, and Hall (2011)), the BiLSTM may still manage to extract ℓ-specific information from the single sentence $x \in \mathbf{x}^{(\ell)}$ that it is parsing.[5] The additional GD training languages apparently help it learn to do so in a way that generalizes to new languages.

To better understand the trend, we study how the performance varies when more

[5]That is, our baseline system has learned a single parser that can handle a cross-linguistic variety of POS sequences (cf. McDonald, Petrov, and Hall, 2011; Ammar et al., 2016, section 4.2), just as the reader was able to parse `VERB DET NOUN ADJ DET NOUN` in Section 1.4.1.

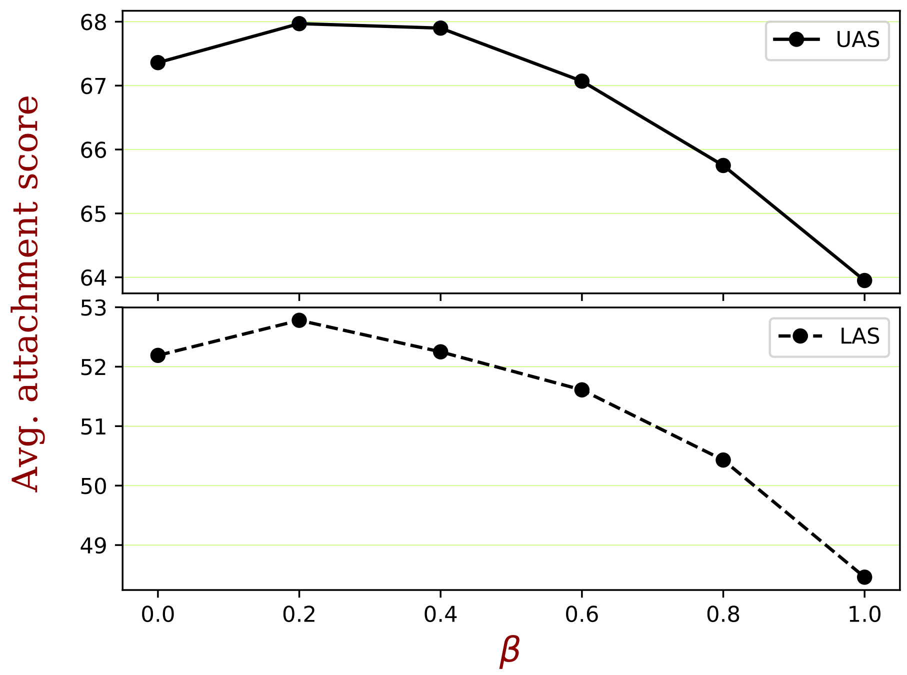


Figure 5.3: Effect of β. The UAS and LAS (y-axis) of the baseline system as a function of β (x-axis).

synthetic languages are used. As shown in Figure 5.3, when $\beta = 1$, all the training languages are sampled from real languages. By gradually increasing the proportion of GD languages (reducing β from Section 5.5.2), the baseline UAS increases dramatically from 63.95 to 67.97. However, if all languages are uniformly sampled ($\beta = \frac{16}{4624+16} \approx 0.003$) or only synthetic languages are used ($\beta = 0$), the UAS falls back slightly to 67.42 or 67.36. The best β value is 0.2, which treats each real language as $\frac{0.2/16}{0.8/4624} \approx 72$ times more helpful than each synthetic language, yet 80% of the training data is contributed by synthetic languages. $\beta = 0.2$ was also optimal for the non-baseline systems in Table 5.1.

Unparsed corpora. The systems that exploit unparsed corpora consistently outperform the baseline system in both the UD and +GD conditions. To investigate, we examine the impact of reducing $\boldsymbol{u}^{(\ell)}$ when parsing a held-out language ℓ. We used

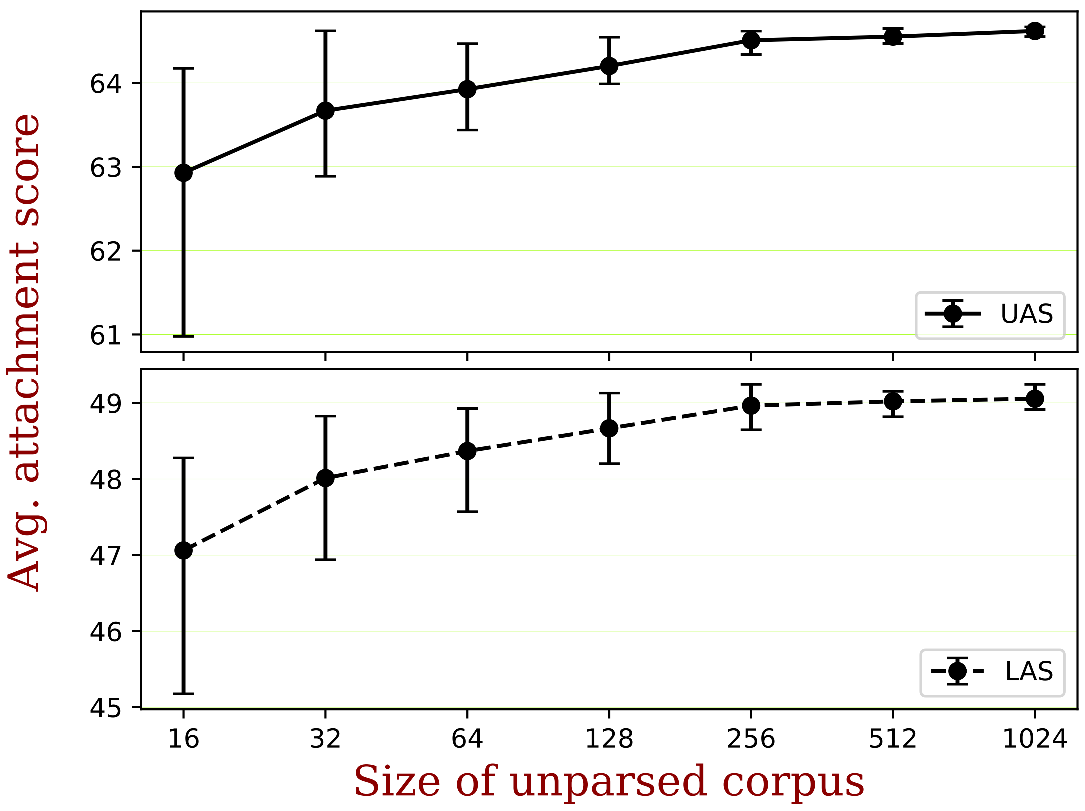


Figure 5.4: Effect of the size $|\boldsymbol{u}^{(\ell)}|$ of the unparsed corpus. The y-axis represents the cross-validation UAS and LAS scores, averaged over the 7 languages that have $|\boldsymbol{u}^{(\ell)}| \geq 9000$ sentences, when using only a subset of the sentences from $\boldsymbol{u}^{(\ell)}$. Using all of $\boldsymbol{u}^{(\ell)}$ would achieve 64.61 UAS and 49.04 LAS. The plot shows the average over 10 runs with different random subsets; the error bars indicate the 10th to the 90th percentile of those runs. The 7 languages are Finnish (Finnic), Norwegian (Germanic), Dutch (Germanic), Czech (Slavic), German (Germanic), Hindi (Indic), and English (Germanic).

the system in row N and column +GD of Table 5.1, which was trained on full-sized $\boldsymbol{u}$ corpora. When testing on a held-out language ℓ, we compute $\mathrm{T}(\boldsymbol{u}^{(\ell)})$ using only a random size-t subset of $\boldsymbol{u}^{(\ell)}$. As shown in Figure 5.4, the system does not need a very large unparsed corpus—most of the benefit is obtained by $t = 256$. Nonetheless, a larger corpus always achieves a better and more stable performance.

5.6.3 Comparison to SST

Besides Baseline, another directly comparable approach is SST (Section 5.2.2). As shown in Table 5.1, SST gives a stronger baseline on the UD column—as good as H+N. However, this advantage does not carry over to the +GD column, meaning that SST cannot exploit the extra training data. Figure 3.5 already found that GD languages provide diminishing benefit to SST as more UD languages get involved.[6] For H+N, however, the extra GD languages do help to identify the truly useful surface patterns in $\boldsymbol{u}$.

We also considered trying model interpolation (Rosa and Žabokrtský, 2015b). Unfortunately, as mentioned in Section 5.2.2, this method is impractical with GD languages, because it requires storing 4624 (Section 5.6.1) additional local models. Nonetheless, we can estimate an "upper bound" on how well the interpolation might do. Our upper bound is SST where an oracle is used to choose the source language; Rosa and Žabokrtský (2015b) found that in practice, this does better than interpolation. This approximate upper bound is 68.03 of UAS and 52.10 of LAS, neither of which is significantly better than H+N on UD, but both of which are significantly outperformed by H+N on +GD.

5.6.4 Oracle typology vs. our learned $\mathrm{T}(\boldsymbol{u})$

The results in Table 5.1 demonstrate that we learned to extract features $\mathrm{T}(\boldsymbol{u})$, from the unparsed target corpus $\boldsymbol{u}$, that improve the baseline parser. We consider replacing $\mathrm{T}(\boldsymbol{u})$ by an oracle that has access to the true syntax of the target language. We consider two different oracles, T_{D} and T_{W}.

[6]The number of real treebanks in our cross-validation setting is 16, greater than the 10 in Chapter 3.

ID	Feature Description	Values
81A	Order of Subject, Object and Verb	SVO, SOV, VSO, VOS, OVS, OSV
82A	Order of Subject and Verb	SV, VS
83A	Order of Object and Verb	OV, VO
85A	Order of Adposition and Noun	Postpositions, Prepositions, Inpositions
86A	Order of Genitive and Noun	Gen-Noun, Noun-Gen
87A	Order of Adjective and Noun	Adj-Noun, Noun-Adj
88A	Order of Demonstrative and Noun	Dem-Noun, Noun-Dem
89A	Order of Numeral and Noun	Num-Noun, Noun-Num

Table 5.2: The WALS features used in our experiment. For each feature, besides the values in the table, we use an additional "ND" for the languages with no dominant order.

$\mathrm{T_D}$ is the *directionalities* typology that was studied by Liu (2010) and used as a training target by Chapter 4. Specifically, $\mathrm{T_D} \in [0,1]^{57}$ is a vector of the directionalities of each type of dependency relation; it specifies what fraction of direct objects fall to the right of the verb, and so on.[7] In principle, this should be very helpful for parsing, but it must be extracted from a treebank, which is presumably unavailable for unknown languages.

We also consider $\mathrm{T_W}$—the WALS features—as the typological classification given by linguists. This resembles the previous multi-language learning approaches (Naseem, Barzilay, and Globerson, 2012; Täckström, McDonald, and Nivre, 2013; Zhang and Barzilay, 2015; Ammar et al., 2016) that exploited the WALS features. As shown in Table 5.2, we use 81A, 82A, 83A, 85A, 86A, 87A, 88A and 89A—a union of WALS features used by those works. In order to derive the WALS features for a synthetic GD language, we first copy the features from its substrate language Section 3.3.1. We then replace the 81A, 82A, 83A features—which concern the order

[7]The directionality of a relation a in language ℓ is given by $\frac{\mathrm{count}_\ell(\xrightarrow{a})}{\mathrm{count}_\ell(a)}$, where $\mathrm{count}_\ell(\xrightarrow{a})$ is the count of a-relations that point from left to right, and $\mathrm{count}_\ell(a)$ is the count of all a-relations.

between verbs and their dependents—by those of its V-superstrate language[8] (if any). We replace 85A, 86A, 87A, 88A and 89A—which concern the order between nouns and their dependents—by those of its N-superstrate language (if any).

As a pleasant surprise, we find that our best system (H+N) is competitive with both oracle methods. It outperforms both of them on both UAS and LAS, and the improvements are significant and substantial in 3 of these 4 cases. Our parser has learned to extract information $\mathrm{T}(\boldsymbol{u})$ that is not only cheap (no treebank needed), but also at least as useful as "gold" typology for parsing.

5.6.5 Selected hyperparameter settings

For the rest of the experiments, we use the H+N system, as it wins under cross-validation on both UD and +GD (Table 5.1). This is a combination via Equation (5.7) of the best H system and the best N system under cross-validation, with the mixture hyperparameter α also chosen by cross-validation.

For both UD and +GD, cross-validation selected 125 as the sizes of the LSTM hidden states and 100 as the sizes of the hidden layers for scoring arcs (the length of $\mathbf{v}$ in equation Equation (5.5)).

Hyperparameters for UD. The H system computes $\mathrm{T}(\boldsymbol{u})$ with a 1-layer network (as in Equation (5.2)), with hidden size $h = 128$ and $\sigma = \tanh$ as the activation function. For the N system, $\mathrm{T}(\boldsymbol{u})$ is a 1-layer network with hidden size $h = 64$ and $\sigma = \text{sigmoid}$ as the activation function. The size of the hidden state of GRU as shown in Figure 4.2 is 128. The mixture weight for the final H+N system is $\alpha = 0.5$.

Hyperparameters for +GD. The H system computes $\mathrm{T}(\boldsymbol{u})$ with a 2-layer network

[8]The language whose word order model is used to permute the dependents of the verbs. See Section 3.3.1 for details.

(as shown in Figure 5.1), with $h = 128$ and $\sigma =$ sigmoid for both hidden layers. For N, $\mathrm{T}(\boldsymbol{u})$ is a 1-layer network with hidden size $h = 64$ and $\sigma =$ sigmoid. The size of the hidden state of GRU is 256. Both H and N set $\beta = 0.2$ (see Section 5.5.2). The mixture weight for the final H+N system is $\alpha = 0.4$.

5.6.6 Performance on noisy tag sequences

We test our trained system in a more realistic scenario where both $\boldsymbol{u}$ and x for held-out languages consist of noisy POS tags rather than gold POS tags. Following Sections 3.7.4 and 4.6.4, at test time, the gold POS tags in a corpus are replaced by a noisy version produced by the RDRPOSTagger (Nguyen et al., 2014) trained on a subset of the original gold-tagged corpus.[9] Figure 5.5 shows a linear relationship between the performance of our best model (H+N with +GD) and the noisiness of the POS tags, which is controlled by altering the amount of training data. With only 100 training sentences, the performance suffers greatly—the UAS drops from 70.65 to 51.57—making it unclear whether our approach will outperform the baselines in Table 5.1 under this noisy setting, which is left for futher work.[10] Nonetheless, even this is comparable to Naseem et al. (2010) on *gold* POS tags, which yields a UAS of 50.00. That system was the first grammar induction approach to exploit knowledge of the distribution of natural languages, and remained state-of-the-art (Noji, Miyao, and Johnson, 2016) until the work of Mareček (2016) and Martínez Alonso et al. (2017).

[9]Another way to get noisy tags, as a reviewer notes, would have been to use a cross-lingual POS tagger designed for low-resource settings (Täckström et al., 2013; Kim et al., 2017).

[10]Tiedemann and Agić (2016) also show that noisy POS tags have a strong impact on single-source tranfer.

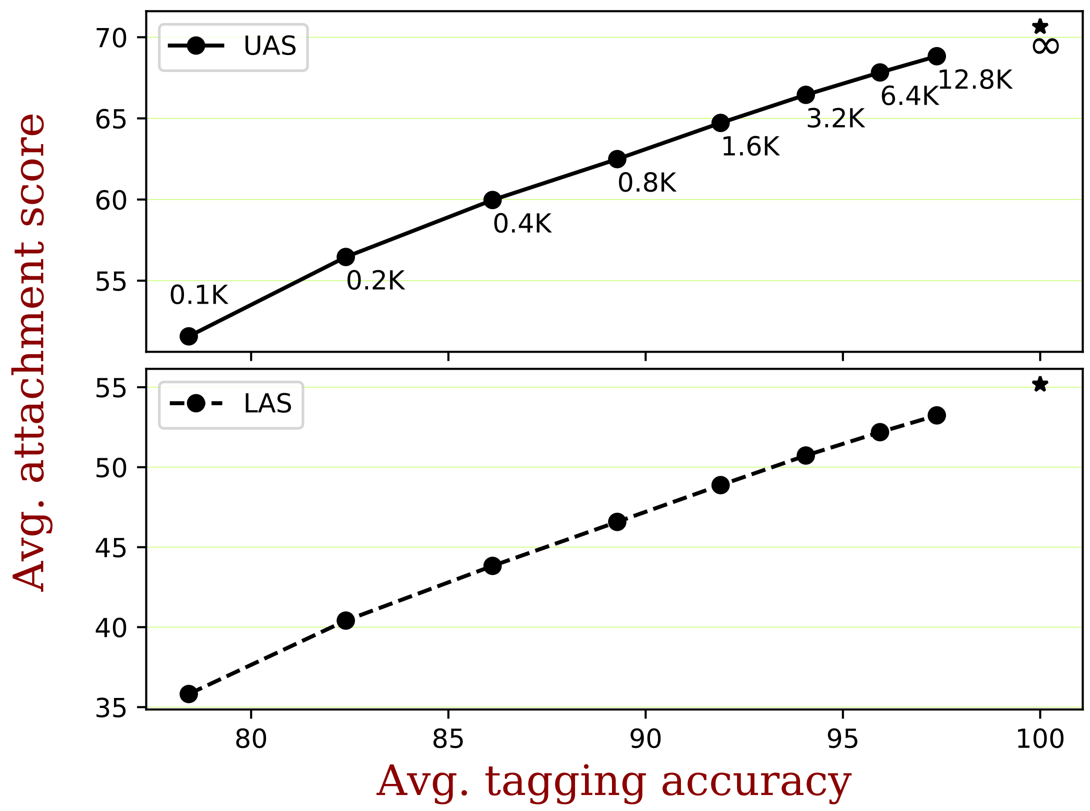


Figure 5.5: Performance on noisy input over 16 training languages. Each dot is an experiment annotated by the number of sentences used to train the tagger. (The rightmost "∞" point uses gold tags instead of a tagger, which is the result from Table 5.1.) The x-axis gives the average accuracy of the trained RDRPOSTagger. The y-axis gives the average parsing performance.

5.6.7 Analysis by dependency relation type

Figure 5.6 breaks down the results by dependency relation type—showing that using $\boldsymbol{u}$ and synthetic data improves results *almost across the board*.

We also notice large differences between labeled and unlabeled F1 scores for some relations, especially rarer ones. In other words, the system mislabels the arcs that it correctly recovers. (Remember from Section 5.6.2 that the hyperparameters were selected to maximize unlabeled scores (UAS) rather than labeled (LAS).)

Figure 5.7 gives the label confusion matrix. While the dark NONE column shows

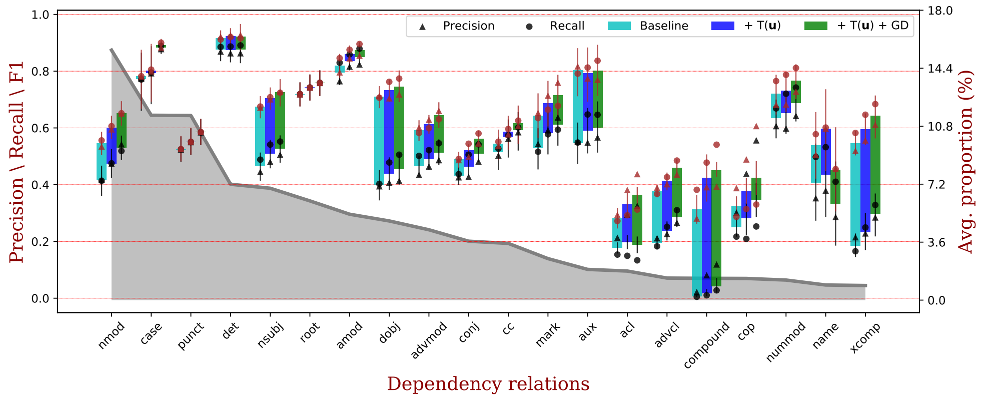


Figure 5.6: Evaluation by dependency relation type, showing an equal-weighted average of the 16 development languages. Each vertical bar spans the range from labeled F1 (bottom) to unlabeled F1 (top), with error bars given by bootstrap resampling over the 16 languages. Precision and recall are also indicated. The pattern is that F1, precision, and recall—both labeled and unlabeled—are improved over baseline when we exploit unlabeled corpora (+T$(\boldsymbol{u})$), and improved again when we augment training data (+T$(\boldsymbol{u})$+GD). The relations are sorted by their average gold proportion in the 16 languages, shown by the gray area and right vertical axis. For example, `nmod` is the most common relation, accounting for 15.5% of all arcs. Altogether, the 20 most frequent relations (shown here) account for 94% of the arcs.

that arcs of each type are often missed altogether (recall errors), the dark diagonal shows that they are usually labeled correctly if found. That said, it is relatively common to confuse the different labels for nominal dependents of verbs (`nsubj`, `dobj`, `nmod`). We suspect that lexical information could help sort out these roles via distributional semantics. Some other mistakes arise from discrepancies in the annotation scheme. For example, `neg` can be easily confused with `advmod`, as some languages (for example, Spanish) use `adv` instead of `part` for negations.

5.6.8 Final evaluation on test data

In all previous sections, we evaluated on the 16 languages in the training set by cross-validation. For the final test, we combine all the 20 treebanks and train the system with the hyperparameters given in Section 5.6.5, then test on the 15 unseen

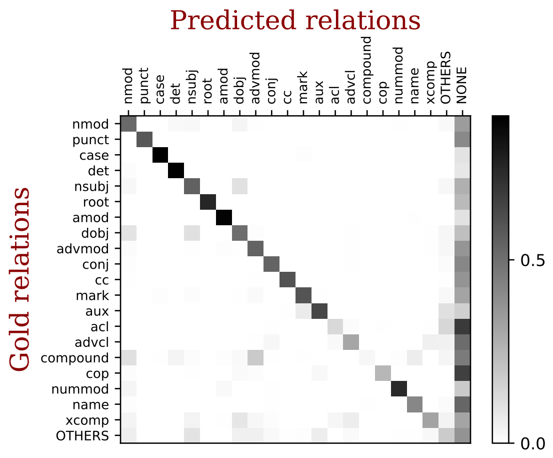


Figure 5.7: The confusion matrix of our parser, as an equal-weight average over 16 development languages. Each row is normalized to sum to 1 and represents a frequent gold relation. For example, the `nsubj` row shows how well we recovered the gold `nsubj` arcs; the (`nsubj`, `dobj`) entry shows $p(\text{predicted} = \texttt{dobj} \mid \text{gold} = \texttt{nsubj})$, which measures the fraction of `nsubj` relations that are recovered but mislabeled as `dobj`. The diagonal represents correct arcs: where dark, it indicates high labeled recall for that relation. The final column represents gold arcs that were not recovered with any label: where dark, it indicates low unlabeled recall for that relation. We show the top 20 relations sorted by gold frequency.

test languages. Table 5.3 displays results on these 15 test languages (top) as well as the cross-validation results on the 16 languages (bottom).

We see that we improve significantly over baseline on almost every language. Indeed, on the test languages, +T$(\boldsymbol{u})$ improves both UAS and LAS by > 3.5 percentage points on average. The improvement grows to > 5.6 if we augment the training data as well (+GD, meaning +T(u)+GD).

One disappointment concerns the *added* benefit on the **LAS** of +GD over just +T$(\boldsymbol{u})$: while this data augmentation helped significantly on nearly every one of the 16 development languages, it produced less consistent improvements on the test languages and hurt some of them. We suspect that this is because we tuned the hyperparameters to maximize UAS, not LAS (Section 5.6.2). As a result, while the *average* benefit across our 15 test languages was fairly large, this sample was not large enough to establish that it was significantly greater from 0, that is, that future test languages would also see an improvement from data augmentation.

We also notice that there seems to be a small difference between the pattern of results on development versus test languages. This may simply reflect overfitting to the development languages, but we also note that the test languages (chosen by Chapter 3) tended to have considerably smaller unparsed corpora $\boldsymbol{u}$, so there may be a domain mismatch problem. To ameliorate this problem, one could include training examples with versions of $\boldsymbol{u}$ that are truncated to lengths seen in test data (cf. Figure 5.4). One could also include the size $|\boldsymbol{u}|$ explicitly in $\mathrm{T}(\boldsymbol{u})$.

5.7 Conclusion and Future Work

We apply our AB inference function to delexicalized dependency parsing that can better parse sentences of an unknown language by exploiting an unparsed (but POS-tagged) corpus of that language. Unlike grammar induction, which estimates a PCFG from the unparsed corpus, we train a neural network to extract a feature vector from the unparsed corpus that helps a subsequent neural parser. By end-to-end training on the treebanks of many languages (optionally including synthetic languages), our neural network can extract linguistic information that helps neural dependency parsing.

Variants of our architecture are possible. In future work, the neural parser could use *attention* to look at individual relevant sentences of $\boldsymbol{u}$, which are posited to be *triggers* in some theories of child grammar acquisition (Gibson and Wexler, 1994; Frank and Kapur, 1996). We could also try injecting $\mathrm{T}(\boldsymbol{u})$ into the neural parser by means other than concatenating it with the input POS embeddings.

	UAS			LAS		
Language	B	+T($\boldsymbol{u}$)	+GD	B	+T($\boldsymbol{u}$)	+GD
Basque	49.89	54.34	**57.59**	27.07	31.46	**35.32**
Croatian	65.03	**67.78**	**68.65**	48.68	52.29	**53.68**
Greek	65.91	68.37	**70.46**	50.1	56.73	**57.89**
Hebrew	62.58	**66.27**	65.3	49.71	**53.29**	52.08
Hungarian	58.5	64.13	**70.02**	42.85	47.73	**49.99**
Indonesian	55.22	**64.63**	**65.36**	39.46	**47.63**	**48.38**
Irish	58.58	61.51	**62.21**	39.06	40.75	**42.36**
Japanese	54.97	**60.41**	58.4	37.57	**40.6**	37.86
Slavonic	**68.79**	**71.13**	**71.54**	40.03	**43.95**	**44.12**
Persian	40.38	34.2	**57.25**	30.06	24.6	**47.14**
Polish	72.15	76.85	**78.28**	50.08	54.85	**58.15**
Romanian	66.55	69.69	**71.18**	50.9	53.42	**55.17**
Slovenian	72.21	76.06	**78.62**	57.09	61.48	**64.1**
Swedish	72.26	**75.32**	**73.89**	55.35	**58.42**	52.39
Tamil	51.59	**57.53**	**57.91**	28.39	**37.81**	32.52
Avg.	60.97	64.55	**67.11**	43.09	**47.00**	**48.74**
Arabic	45.75	49.32	**53.83**	36.4	40.39	**44.14**
Danish	66.71	**68.41**	**68.4**	52.24	**54.49**	**54.67**
Norwegian	68.35	**70.89**	**71.22**	52.33	**56.01**	**56.37**
Czech	64.31	68.77	**72.42**	50.19	55.16	**57.95**
Estonian	72.67	79.88	**81.67**	42.81	51.32	**52.57**
Portuguese	70.48	73.47	**74.83**	60.85	63.18	**64.96**
German	62.18	63.62	**66.52**	48.44	49.46	**53.51**
Gothic	63.23	66.72	**70.75**	39.1	42.6	**45.17**
Italian	75.9	79.24	**80.57**	65.46	68.8	**70.0**
Bulgarian	77.57	79.53	**83.66**	55.83	57.65	**61.47**
Finnish	53.73	58.03	**60.44**	34.68	39.55	**43.15**
French	74.57	76.88	**79.34**	64.1	66.83	**68.48**
Dutch	59.63	**62.58**	60.31	45.84	**48.28**	47.98
English	61.66	63.99	**65.9**	47.61	51.43	**53.13**
Hindi	35.84	40.74	**62.45**	18.63	21.65	**41.12**
Spanish	70.65	75.36	**78.03**	60.8	65.45	**68.23**
All Avg.	62.51	65.99	**68.94**	45.86	49.59	**52.07**

Table 5.3: Data splits and final evaluation on the 15 test languages (top), along with cross-validation results on the 16 development languages (bottom) grouped by 5 folds (separated by dashed lines). For languages with multiple treebanks, we identify them by subscripts. We use "Slavonic" for Old Church Slavonic. Column B is the baseline that doesn't use $\mathrm{T}(\boldsymbol{u})$ (McDonald, Petrov, and Hall, 2011). $+\mathrm{T}(\boldsymbol{u})$ is our H+N system, and +GD is that system when the training data is augmented with synthetic languages. In comparing among these three systems, we boldface the highest score as well as all scores that are not significantly worse (paired permutation test, $p < 0.05$). If a row is an average over many sentences of a single language, then each paired datapoint is a *sentence*, so a significant improvement should generalize to new sentences. But if a row is an average, then each paired datapoint is a *language* (as in Table 5.1), so a significant improvement should generalize to new languages.

We might also consider parsing architectures other than BIST, such as the LSTM-Minus architecture for scoring spans (Cross and Huang, 2016), or the recent attention-based arc-factored model (Dozat and Manning, 2017). Finally, our approach is applicable to tasks other than dependency parsing, such as constituent parsing or semantic parsing—if suitable treebanks are available for many training languages.

For applied uses, it would be interesting to combine the *unsupervised* techniques of this work with *low-resource* techniques that make use of some annotated or parallel data in the target language. It would also be interesting to include further synthetic languages that have been modified to better resemble the actual target languages, using the method of Chapter 6.

It is important to relax the delexicalized assumption. As shown in Section 5.6.6, the performance of our system relies heavily on the gold POS tags, which are presumably not available for unknown languages. What is available is lexical information—which has proved to be very important for supervised parsing, and should help unsupervised parsers as well. As discussed in Section 5.6.7, some errors seem easily fixable by considering word distributions. In the future, we will explore ways to extend our cross-linguistic parser to work with word sequences rather than POS sequences, perhaps by learning a cross-language word representation that is shared among training and test languages (Ruder, Vulić, and Søgaard, 2017).

Chapter 6

Synthetic Data Made to Order

While Chapter 3 introduced to mix-and-match the source languages to generate synthetic languages so that they are diverse enough, Section 3.8 [point 5] asks whether the synthesizing process could be biased directly to a given target language. In this chapter, we extend this discussion and propose such an *on-demand* approach by (stochastically) permuting source dependency treebank so that its surface part-of-speech statistics approximately match those of the target language. The parameters of the permutation model can be evaluated for quality by dynamic programming and tuned by gradient descent (up to a local optimum). This optimization procedure yields trees for a new artificial language that resembles the target language. We show that delexicalized parsers for the target language can be successfully trained using such "made to order" artificial languages.

6.1 Introduction

We begin with the observation in Figure 3.4 on page 56, where the clusters formed by the real languages indicates a strong correlation between the closeness (in terms of the language family) and the transfer parsing (train on one language and parse on another) performance. In other words, to approximately parse an unfamiliar language, it helps to have a treebank of a similar language. Thanks to the recent development of multi-lingual treebanks (McDonald et al., 2013; Nivre et al., 2015; Nivre et al., 2019), finding such similar language(s) is increasingly possible. The idea is to parse the sentences of the target language with a supervised parser trained on the treebanks of one or more closely related languages (e.g., using German to parse Luxembourgish). Although the parser cannot be expected to know the words of the target language, it can make do with POS tags (McDonald, Petrov, and Hall, 2011; Täckström, McDonald, and Nivre, 2013; Zhang and Barzilay, 2015) or cross-lingual word embeddings (Duong et al., 2015b; Guo et al., 2016; Ammar et al., 2016).

If the name of the target language is known, one could retrieve its family or typological information from some existing resources such as WALS (Dryer and Haspelmath, 2013) to find some languages that are linguistically related for training Smith et al. (2018). If the language name is unknown, one could use the unsupervised selection approach introduced in Section 3.7.2, where the closeness is measured by the distance between POS language models trained on the source and target corpora. To improve this approach, one straightforward (traditional) way is enriching the pool of real languages to increase the odds of related languages—as the black curve in Figure 3.5 shows. However, the annotation of real languages requires expensive human effort which may not be cost-effective for improving parsing performance. Therefore, both Chapter 3 and this chapter are focusing on synthesizing languages.

6.1.1 Chapter 3: Universal and reusable synthetic data

Chapter 3 introduces the Galactic Dependencies (GD)—a large and diverse collection of synthetic languages generated from an automatic mix-and-match mechanism over the real languages, which extends the learning curve in Figure 3.5 by increasing the *base number* of source languages to fill in the gap between the real source languages and the target language (Figure 3.4). One notable characteristic of this approach is the generation of synthetic data fully depends on the given real languages. In other words, the synthetic data will be stored and reused for any target languages.

6.1.2 This chapter: Tailored synthetic data

The novelty of this chapter is that instead of increasing the number of synthetic languages, we improve the *relevance* of the synthetic languages by "personalizing" the permutation of a real language toward the target language. How? Given a dependency treebank of a possibly distant source language, we stochastically permute the children of each node, according to some distribution that makes the permuted language close to the target language.

And how do we find this distribution? We adopt the tree-permutation model described in Section 3.4, and design a dynamic programming algorithm which, for any given distribution p in Section 3.4's family, can compute the *expected counts* of all POS bigrams in the permuted source treebank. This allows us to evaluate p by computing the divergence between the bigram POS language model formed by these expected counts, and the one formed by the *observed counts* of POS bigrams in the unparsed target language. In order to find a p that locally minimizes this divergence, we adjust the model parameters by stochastic gradient descent (SGD). In contrast to Chapter 3, the syntactic data will be generated *on-the-fly* after seeing the target language.

6.1.3 Key limitations

Better measures of surface closeness between two languages might be devised. However, even counting the expected POS N-grams is moderately expensive, taking time exponential in N if done exactly. So we compute only these local statistics, and only for $N = 2$. We certainly need $N > 1$ because the 1-gram distribution is not affected by permutation at all. $N = 2$ captures useful bigram statistics: for example, to mimic a verb-final language with prenominal modifiers, we would seek constituent permutations that result in matching its relatively high rate of `VERB`–`PUNCT` and `ADJ`–`NOUN` bigrams. While $N > 2$ might have improved the results, it was too slow for our large-scale experimental design. Section 6.7 discusses how richer measures could be used in the future.

Again, throughout this chapter, we assume that our corpora are annotated with gold POS tags, even in the target language (which lacks any gold training trees). Section 6.7 discusses a possible avenue for doing without gold tags.

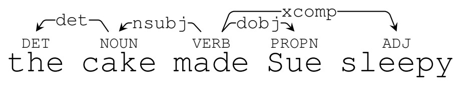
6.2 Modeling Surface Realization

We motivate our approach with a different view on the idea of tree permutation described in Section 3.4. Let us suppose that the dependency tree for a sentence starts as a labeled graph—a tree in which siblings are not yet ordered with respect to their parent or one another. Each language has some systematic way to *realize* its unordered trees as surface strings:[1] it imposes a particular order on the tree's word tokens. More precisely, a language specifies a *distribution* $p(\text{string} \mid \text{unordered tree})$ over a tree's possible realizations.

As an engineering matter, we now make the strong assumption that the unordered dependency trees are similar across languages. That is, we suppose that different languages use similar underlying syntactic/semantic graphs, but differ in how they realize this graph structure on the surface.

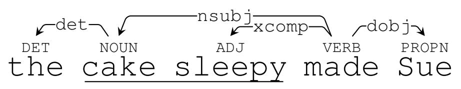
Thus, given a gold POS corpus $\boldsymbol{u}$ of the unknown target language, we may hope to explain its distribution of surface POS bigrams as the result of applying some target-language surface realization model to the distribution of cross-linguistically "typical" unordered trees. To obtain samples of the latter distribution, we use the treebanks of one or more *other* languages. The present work evaluates the method when only a single source treebank is used. In the future, we could try tuning a mixture of all available source treebanks.

6.2.1 Realization is systematic

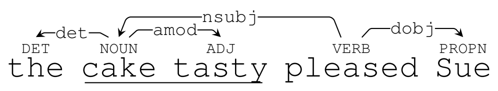
We presume that the target language applies the same stochastic realization model to all trees. All that we can optimize is the parameter vector of this model. Thus, we deny ourselves the freedom to realize each individual tree in an *ad hoc* way. To see why this is important, suppose the target language is French, whose corpus $\boldsymbol{u}$ contains many `NOUN–ADJ` bigrams. We could achieve such a bigram from the unordered source tree 

by ordering it to yield

[1]Modeling this process was the topic of the recent Surface Realization Shared Task (Mille et al., 2018). Most relevant is work on *tree linearization* (Filippova and Strube, 2009; Futrell and Gibson, 2015; Puzikov and Gurevych, 2018).



. However, that realization is not in fact appropriate for French, so that ordered tree would not be a useful training tree for French. Our approach should disprefer this tempting but incorrect realization, because any model with a high probability of this realization would, if applied *systematically* over the whole corpus, also yield sentences like `He sleepy made Sue`, with unwanted `PRON`–`ADJ` bigrams that would not match the surface statistics of French. We hope our approach will instead choose the realization model that is correct for French, in which the `NOUN`–`ADJ` bigrams arise instead from source trees where the `ADJ` is a dependent of the `NOUN`, yielding (e.g.)



. This has the same POS sequence as the example above (as it happens), but now assigns the correct tree to it.

6.2.2 A parametric realization model

As our family of realization distributions, we apply the log-linear model in Section 3.4, which assumes that the root node a of the unordered dependency tree selects an ordering $o(a)$ of the n_a nodes consisting of a and its $n_a - 1$ dependent children. The procedure is repeated recursively at the child nodes. This method can produce only projective trees.

Each node a draws its ordering $o(a)$ independently according to

$$p_{\boldsymbol{\theta}}(o \mid a) = \frac{1}{Z(a)} \exp \sum_{1 \le i < j \le n_a} \boldsymbol{\theta} \cdot \mathbf{f}(o, i, j) \tag{6.1}$$

which is a distribution over the $n_a!$ possible orderings. $Z(a)$ is a normalizing constant. $\mathbf{f}$ is a feature vector extracted from the ordered pair of nodes o_i, o_j, and $\boldsymbol{\theta}$ is the model's parameter vector of feature weights.

To construct the feature vector $\mathbf{f}(o, i, j)$, we use the following subset of the feature templates of Section 3.4.4.

- $\texttt{L.}t_i\texttt{.}r_i$, provided that $r_j = \texttt{head}$.
- $\texttt{L.}t_i\texttt{.}r_i\texttt{.}t_j\texttt{.}r_j$, provided that $r_i \neq \texttt{head}$ and $r_j \neq \texttt{head}$.
- $d.t_i.r_i.t_j.r_j$, where d is `l` (left), `m` (middle), or `r` (right) according to whether the head position h satisfies $i < j < h$, $i < h < j$, or $h < i < j$.
- $\texttt{A.}t_i\texttt{.}r_i\texttt{.}t_j\texttt{.}r_j$, provided that $j = i + 1$.

These templates are instantiated with all tags and relations that appear in the source treebank. In contrast to Chapter 3, the ordering model that we tune on the source treebank is never applied to any other treebank. Thus, there is no need to include tags or relations that do not appear in the source treebank, nor do we need the *backoff* features of Chapter 3. Also, for speed, we exclude the "high-order" features from that paper. Following Section 3.3.1, we choose new orderings for the noun and verb nodes only, preserving the source treebank's order at all other nodes a.

6.2.3 Generating training data

Given a source treebank $\boldsymbol{y}$ and some parameters $\boldsymbol{\theta}$, we can use Equation (6.1) to randomly sample realizations of the trees in $\boldsymbol{y}$. The effect is to reorder dependent phrases within those trees. The resulting permuted treebank $\boldsymbol{y}'$ can be used to train a parser for the target language.

6.2.4 Choosing parameters θ

So how do we choose $\boldsymbol{\theta}$ that works for the target language? Suppose $\boldsymbol{u}$ is a corpus of target-language POS sequences, using the same set of POS tags as $\boldsymbol{y}$. We evaluate parameters $\boldsymbol{\theta}$ according to whether POS tag sequences in $\boldsymbol{y}'$ will be distributed like POS tag sequences in $\boldsymbol{u}$.

To do this, first we estimate a bigram language model $\hat{q}$ from the actual distribution q of POS sequences observed in $\boldsymbol{u}$. Second, let $p_{\boldsymbol{\theta}}$ denote the distribution of POS sequences that we expect to see in $\boldsymbol{y}'$, that is, POS sequences obtained by stochastically realizing observed trees in $\boldsymbol{y}$ according to $\boldsymbol{\theta}$. We estimate another bigram model $\hat{p}_{\boldsymbol{\theta}}$ from this distribution $p_{\boldsymbol{\theta}}$.

We then try to set $\boldsymbol{\theta}$, using SGD, to minimize a divergence $D(\hat{p}_{\boldsymbol{\theta}}, \hat{q})$ that we will define below.

6.2.4.1 Estimation of bigram models

Estimating $\hat{q}$ is straightforward: $\hat{q}(t \mid s) = c_q(st)/c_q(s)$, where $c_q(st)$ is the count of POS bigram st in the average[2] sentence of $\boldsymbol{u}$ and $c_q(s) = \sum_{t'} c_q(st')$. We estimate $\hat{p}_{\boldsymbol{\theta}}$ in the same way, where $c_p(st)$ denotes the *expected* count of st in a random POS sequence $\mathbf{s} \sim p_{\boldsymbol{\theta}}$. This is equivalent to choosing $\hat{q}, \hat{p}_{\boldsymbol{\theta}}$ to minimize the KL-divergences $\mathrm{KL}(q \mid\mid \hat{q}), \mathrm{KL}(p_{\boldsymbol{\theta}} \mid\mid \hat{p}_{\boldsymbol{\theta}})$. It ensures that each model's expected bigram counts match those in the POS sequences.

However, these maximum-likelihood estimates might overfit on our finite data, $\boldsymbol{u}$ and $\boldsymbol{y}$. We therefore smooth both models by first adding $\lambda = 0.1$ to all bigram counts

[2] A more familiar definition of c_q would use the *total* count in $\boldsymbol{u}$. Our definition, which yields the same bigram probabilities, is analogous to our definition of c_p. This c_p is needed for $\mathrm{KL}(p \mid\mid q)$ in Equation (6.3), and c_q symmetrically for $\mathrm{KL}(q \mid\mid p)$.

$c_q(st)$ and $c_p(st)$.[3]

6.2.4.2 Divergence of bigram models

We need a metric to evaluate $\boldsymbol{\theta}$. If p and q are bigram language models over POS sequences $\mathbf{s}$ (sentences), their Kullback-Leibler divergence is

$$\mathrm{KL}(p \mid\mid q) \overset{\text{def}}{=} \mathop{\mathbb{E}}_{\mathbf{s}\sim p}[\log p(\mathbf{s}) - \log q(\mathbf{s})] \tag{6.2}$$

$$= \sum_{s,t} c_p(st) \cdot (\log p(t \mid s) - \log q(t \mid s)) \tag{6.3}$$

where $\mathbf{s}$ ranges over POS sequences and st ranges over POS bigrams. These include bigrams where $s = \texttt{BOS}$ or $t = \texttt{EOS}$, which are boundary tags that we take to surround $\mathbf{s}$.

All quantities in Equation (6.3) can be determined directly from the (expected) bigram counts given by c_p and c_q. No other model estimation is needed.

A concern about Equation (6.3) is that a single bigram st that is badly under-represented in q may contribute an arbitrarily large term $\log \frac{p(t|s)}{q(t|s)}$. To limit this contribution to at most $\log \frac{1}{\alpha}$, for some small $\alpha \in (0, 1)$, we define $\mathrm{KL}_\alpha(p \mid\mid q)$ by a variant of Equation (6.3) in which $q(t \mid s)$ has been replaced by $\tilde{q}(t \mid s) \overset{\text{def}}{=} \alpha p(t \mid s) + (1-\alpha) q(t \mid s)$.[4]

Our final divergence metric $D(\hat{p}_{\boldsymbol{\theta}}, \hat{q})$ defines D as a linear combination of exclusive and inclusive KL_α divergences, which respectively emphasize $p_{\boldsymbol{\theta}}$'s precision and

[3]Ideally one should tune λ to minimize the language model perplexity on held-out data (e.g., by cross-validation).

[4]This is inspired by the α-skew divergence of Lee (1999) and Lee (2001). Indeed, we may regard $\mathrm{KL}_\alpha(p \mid\mid q)$ as the α-skew divergence between the unigram distributions $p(\cdot \mid s)$ and $q(\cdot \mid s)$, averaged over all s in proportion to $c_p(s)$. In principle, we could have used the α-skew divergence between the distributions $p(\cdot)$ and $q(\cdot)$ over POS sequences $\mathbf{s}$, but computing that would have required a sampling-based approximation (Section 6.7).

recall at matching q's bigrams:

$$D(p,q) = (1-\beta) \cdot \frac{\mathrm{KL}_{\alpha_1}(p \,||\, q)}{\mathbb{E}_{\mathbf{s} \sim p}[\,|\mathbf{s}|\,]} + \beta \cdot \frac{\mathrm{KL}_{\alpha_2}(q \,||\, p)}{\mathbb{E}_{\mathbf{s} \sim q}[\,|\mathbf{s}|\,]} \tag{6.4}$$

where $\beta, \alpha_1, \alpha_2$ are tuned by cross-validation to maximize the downstream parsing performance. The division by average sentence length converts KL from nats per sentence to nats per word,[5] so that the KL values have comparable scale even if $\boldsymbol{y}$ has much longer or shorter sentences than $\boldsymbol{u}$.

6.3 Algorithms

6.3.1 Efficiently computing expected counts

We now present Algorithm 6—a polynomial-time algorithm for computing the expected bigram counts c_p under $p_{\boldsymbol{\theta}}$ (or equivalently $\hat{p}_{\boldsymbol{\theta}}$), for use above. This averages expected counts from each unordered tree $\mathbf{t} \in \boldsymbol{y}$. The insight of this algorithm is that rather than sampling a single realization of $\mathbf{t}$ (as $\boldsymbol{y}'$ does), we can use dynamic programming to sum efficiently over all of its exponentially many realizations. This gives an exact answer. It algorithmically resembles tree-to-string machine translation, which likewise considers the possible reorderings of a source tree and incorporates a language model by similarly tracking their surface N-grams (Chiang, 2007, Section 5.3.2).

For each node a of the tree $\mathbf{t}$, let the POS string $\mathbf{s}_a$ be the realization of the subtree rooted at a. Let $c_a(st)$ be the expected count of bigram st in $\mathbf{s}_a$, whose distribution is governed by Equation (6.1). We allow $s = \texttt{BOS}$ or $t = \texttt{EOS}$ as defined in Section 6.2.4.2.

[5]Recall that the units of negated log-probability are called bits for log base 2, but nats for log base e.

Algorithm 6 A recursive routine for computing the expected bigram counts c_a from $p_{\boldsymbol{\theta}}$. c_{root} is the c_p function needed by Section 6.2.4. LAZYCOMPUTE (Algorithm 7) is a subroutine for efficiently computing the expected node bigram counts $p_a(i, j)$, which will be described in Section 6.3.2.

Input: A node a in the dependency tree; current model parameters $\boldsymbol{\theta}$
Output: Sparse map c_a where $c_a[st]$ gives the expected count $c_a(st)$ for each POS bigram st

```
 1: procedure ECOUNTNODE($a, \boldsymbol{\theta}$)
 2:     $a_0 = \texttt{BOS}$; $(a_1, \ldots, a_{n-1}) = \text{children}(a)$; $a_n = \text{head}(a)$; $a_{n+1} = \texttt{EOS}$    ▷ $\vec{a}$ is the node sequence defined in Section 6.3.1
 3:     $c_a \leftarrow \{\,\}$    ▷ map we're constructing, initialized to empty; undefined count $c_a[st]$ can be interpreted as 0
 4:     for $i = 1$ to $n - 1$ do
 5:         $c_{a_i} \leftarrow \text{ECOUNTNODE}(a_i)$    ▷ recursively compute expected counts for any subtrees rooted at children($a$)
 6:     $c_{a_n} \leftarrow \{\texttt{BOS}\, h \mapsto 1, h\, \texttt{EOS} \mapsto 1\}$ where $h = \text{POS}(\text{head}(a))$    ▷ serves as the base case of the recursive routine
 7:     $c_{a_0} \leftarrow \{\texttt{BOS}\,\texttt{EOS} \mapsto 1\}$    ▷ dummy boundary nodes
 8:     $c_{a_{n+1}} \leftarrow \{\texttt{BOS}\,\texttt{EOS} \mapsto 1\}$
 9:     $p_a \leftarrow \text{LAZYCOMPUTE}(\vec{a}, \boldsymbol{\theta})$    ▷ call Algorithm 7 for node bigram probs $p_a$ (as defined above Equation (6.5))
10:     for $i = 1$ to $n$ do
11:         for $st \in \text{keys}(c_{a_i})$ such that $s \neq \texttt{BOS}, t \neq \texttt{EOS}$ do
12:             $c_a[st] \mathrel{+}= c_{a_i}[st]$    ▷ increase $c_a[st]$ by $c_a^{within}[st]$ using Equation (6.5)
13:     for $i = 0$ to $n$ do
14:         for $j = 1$ to $n + 1$ such that $j \neq i$ do
15:             for $s, t$ such that $s\,\texttt{EOS} \in \text{keys}(c_{a_i})$ and $\texttt{BOS}\, t \in \text{keys}(c_{a_j})$ do
16:                 $c_a[st] \mathrel{+}= p_a[i, j] \cdot c_{a_i}[s\,\texttt{EOS}] \cdot c_{a_j}[\texttt{BOS}\, t]$    ▷ increase $c_a[st]$ by $c_a^{across}[st]$ using Equation (6.5)
17:     return $c_a$
```

The c_a function can be represented as a sparse map from POS bigrams to reals. We compute c_a at each node a of $\mathbf{t}$ in a bottom-up order. The final step computes c_{root}, giving the expected bigram counts in $\mathbf{t}$'s realization $\mathbf{s}$ (that is, c_p in Section 6.2.4).

We find c_a as follows. Let $n = n_a$ and recall from Section 6.2.2 that $o(a)$ is an ordering of $a_1, \ldots, a_n$, where $a_1, \ldots, a_{n-1}$ are the child nodes of a, and a_n is a dummy node representing a's head token. Also, let a_0 and a_{n+1} be dummy nodes that always appear at the start and end of any ordering.

For all $0 \leq i \leq n$ and $1 \leq j \leq n+1$, let $p_a(i,j)$ denote the expected count of the $a_i a_j$ node bigram—the probability that $o(a)$ places node a_i immediately before node a_j. These *node bigram probabilities* can be obtained by enumerating all possible orderings o, a matter we return to below.

It is now easy to compute c_a:

$$c_a(st) = c_a^{\text{within}}(st) + c_a^{\text{between}}(st) \tag{6.5}$$

$$c_a^{\text{within}}(st) = \begin{cases} \sum_{i=1}^{n} c_{a_i}(st) & \text{if } s \neq \texttt{BOS}, t \neq \texttt{EOS} \\ 0 & \text{otherwise} \end{cases}$$

$$c_a^{\text{across}}(st) = \sum_{i=0}^{n} \sum_{j=1}^{n+1} p_a(i,j) c_{a_i}(s\,\texttt{EOS}) c_{a_j}(\texttt{BOS}\,t)$$

That is, c_a inherits all non-boundary bigrams st that fall within its child constituents (via c_a^{within}). It also counts bigrams st that cross the boundary between consecutive nodes (via c_a^{across}), where nodes a_i and a_j are consecutive with probability $p_a(i,j)$.

When computing c_a via Equation (6.5), we will have already computed $c_{a_1}, \ldots, c_{a_{n-1}}$ bottom-up. As for the dummy nodes, a_n is realized by the length-1 string h where h is the head token of node a, while a_0 and a_{n+1} are each realized by the empty string.

Thus, c_{a_n} simply assigns count 1 to the bigrams $\texttt{BOS}\,h$ and $h\,\texttt{EOS}$, and c_{a_0} and $c_{a_{n+1}}$ each assign expected count 1 to $\texttt{BOS EOS}$. (Notice that thus, $c_a^{\text{across}}(st)$ counts $\mathbf{s}_a$'s boundary bigrams—the bigrams st where $s = \texttt{BOS}$ or $t = \texttt{EOS}$—when $i = 0$ or $j = n + 1$ respectively.)

6.3.2 Efficient enumeration over permutations

The main challenge above is computing the node bigram probabilities $p_a(i, j)$. These are marginals of $p(o \mid a)$ as defined by Equation (6.1), which unfortunately is intractable to marginalize: there is no better way than enumerating all $n!$ permutations.

Similar to Section 3.4.1, we used the Steinhaus-Johnson-Trotter (SJT) algorithm (Sedgewick, 1977), which obtains each permutation by applying a single swap to the previous one. Only the features that are affected by this swap need to be recomputed.

Furthermore, the single swap of adjacent nodes only changes 3 bigrams (possibly including boundary bigrams). As a result, it is possible to obtain the marginal probabilities with $O(1)$ additional work per permutation. Pseudocode is given in Algorithm 7. The key is that UPDATE is called when a bigram is about to be destroyed; it increments the bigram's unnormalized probability by the cumulative change to the running total $Z(a)$ since that bigram was last created. Each enumerated permutation swaps two adjacent nodes (thanks to SJT) in the previous permutation. This destroys 3 bigrams, so it first calls UPDATE on those (Line 15–17).

When we train the parameters $\boldsymbol{\theta}$ (Section 6.2.4), we must back-propagate through the whole computation of Equation (6.4), which depends on tag bigram counts $c_a(st)$, which depend via Equation (6.5) on expected node bigram counts $p_a(i, j)$, which depend via Algorithm 7 on the permutation probabilities $p(o \mid a)$, which depend via

Algorithm 7 Computing node bigram probabilities.

Input: Sequence of nodes $\vec{a} = (a_1, \ldots, a_n)$; current model parameters $\boldsymbol{\theta}$

Output: Array p where $p[i,j]$ = marginal probability of node bigram $a_i a_j$ for all $0 \le i < n+1, 0 < j \le n+1$ with $j \ne i$

```
 1: procedure LazyCompute($\vec{a}, \boldsymbol{\theta}$)
 2:     $p \leftarrow \mathbf{0}$    ▷ initialize all marginal bigram probabilities to zero
 3:     $t \leftarrow 0$    ▷ number of permutations considered so far
 4:     $Z^{(t)} \leftarrow 0$    ▷ $Z^{(t)}$ is always total unnormalized probability of first $t$ permutations
 5:     $\rho_i \leftarrow t$ for $0 \le i < n+1$    ▷ $\rho_i$ is the latest permutation at which bigram $(o_i, o_{i+1})$ was not yet adjacent
 6:     $o \leftarrow (1, 2, \ldots, n)$    ▷ initialize $o$ to be identity permutation, $(\forall i) o_i = i$
 7:     procedure Update($i$)
 8:         ▷ This procedure updates the unnormalized marginal probability of the bigram $(o_i, o_{i+1})$, which is about to change
 9:         $p[o_i, o_{i+1}] \mathrel{+}= Z^{(t)} - Z^{(\rho_i)}$    ▷ total partial sum of $Z(a)$ since $(o_i, o_{i+1})$ acquired its current value
10:         $\rho_i \leftarrow t$    ▷ current time is last time at which $(o_i, o_{i+1})$ will have its current value (until later)
11:     $w \leftarrow \boldsymbol{\theta} \cdot \sum_{1 \le i < j \le n} \mathbf{f}(o, i, j)$    ▷ unnormalized log-probability of $o$ from Equation (6.1)
12:     $t \leftarrow t + 1; Z^{(t)} \leftarrow Z^{(t-1)} + \exp w$    ▷ add the first permutation's unnormalized prob into $Z$
13:     ▷ SJT iterates over a sequence of $n! - 1$ swaps, to get the remaining permutations
14:     for $k$ in SJT($n$) do    ▷ here $1 \le k < n$, meaning to swap $(o_k, o_{k+1})$
15:         Update($k-1$)    ▷ increment prob of current bigram $(o_{k-1}, o_k)$ before that bigram goes away
16:         Update($k$)    ▷ similarly for $(o_k, o_{k+1})$
17:         Update($k+1$)    ▷ similarly for $(o_{k+1}, o_{k+2})$
18:         Swap($o_k, o_{k+1}$)
19:         ▷ Update $w$ from Algorithm 7 using only the difference of feature vectors, which is sparse and computable in $O(n)$ time
20:         $w \leftarrow w + \boldsymbol{\theta} \cdot \sum_{1 \le i < j \le n} (\mathbf{f}(o, i, j) - \mathbf{f}(o_{\text{old}}, i, j))$    ▷ where $o_{old}$ is the pre-swap $\boldsymbol{\theta}$ and is similar to $\boldsymbol{\theta}$
21:         $t \leftarrow t + 1; Z^{(t)} \leftarrow Z^{(t-1)} + \exp w$    ▷ add the new permutation's unnormalized prob into $Z$ (same as Algorithm 7)
22:     for $i = 1$ to $n$ do    ▷ count all bigrams in final permutation as we move on from it
23:         Update($i$)
24:     for $i = 0$ to $n$ do
25:         for $j = 1$ to $n+1$ such that $j \ne i$ do
26:             $p[i,j] \leftarrow \frac{p[i,j]}{Z^{(t)}}$    ▷ normalize the probabilities
27:     return the array $p$
```

Equation (6.1) on the feature weights $\boldsymbol{\theta}$.

6.4 Heuristics

6.4.1 Pruning high-degree trees

As a further speedup, we only train on trees with number of words < 40 and $\max_a n_a \leq 5$, so $n_a! \leq 120$.[6] We then produce the synthetic treebank $\boldsymbol{y}'$ (Section 6.2.3) by drawing a single realization of each tree in $\boldsymbol{y}$ for which $\max_a n_a \leq 7$ (following Section 3.4.1). This requires sampling from up to $7! = 5040$ candidates per node, again using SJT.[7]

That is, we run exact algorithms (Section 6.3), but only on a subset of $\boldsymbol{y}$. The subset is not necessarily representative. An improvement would use importance sampling, with a proposal distribution that samples the slower trees less often during SGD but upweights them to compensate.

Section 6.7 suggests a future strategy that would run on all trees in $\boldsymbol{y}$ via approximate, sampling-based algorithms. The exact methods would remain useful for calibrating the approximation quality.

6.4.2 Minibatch estimation of c_p

To minimize Equation (6.4), we use the Adam variant of SGD (Kingma and Ba, 2015), with learning rate 0.01 chosen by cross-validation (Section 6.5.1).

[6]We found that this threshold worked much better than ≤ 4 and about as well as the much slower ≤ 6.

[7]This pruning heuristic retains 36.1% of the trees (averaging over the 20 development treebanks (Section 6.5.1)) for training, and 66.6% for actual realization.

SGD requires a stochastic estimate of the gradient of the training objective. Ordinarily this is done by replacing an expectation over the entire training set with an expectation over a minibatch.

Equation (6.2) with $p = \hat{p}_{\boldsymbol{\theta}}$ is indeed an expectation over sentences of $\boldsymbol{y}$. It can be stochastically estimated as Equation (6.3) where c_p gives the expected bigram counts averaged over only the sentences in a minibatch of $\boldsymbol{y}$. These are found using Section 6.3's algorithms with the current $\boldsymbol{\theta}$. Unfortunately, the term $\log p(t \mid s)$ depends on bigram counts that should be derived from the *entire* corpus $\boldsymbol{y}$ in the same way. Our solution is to simply reuse the *minibatch* estimate of c_p for the latter counts. We use a large minibatch of 500 sentences from $\boldsymbol{y}$ so that this *drop-in estimate* does not introduce too much bias into the stochastic gradient: after all, we only need to estimate bigram statistics on 17 POS types.[8]

By contrast, the c_q values that are used for the expectation in the second term of Equation (6.4) and in $\log q(t \mid s)$ do not change during optimization, so we simply compute them once from all of $\boldsymbol{u}$.

6.4.3 Informed initialization

Unfortunately the objective Equation (6.4) is not convex, so the optimizer is sensitive to initialization (see Section 6.5.4 below for empirical discussion). Initializing $\boldsymbol{\theta} = \mathbf{0}$ (so that $p(o \mid a)$ is uniform) gave poor results in pilot experiments. Instead, we initially choose $\boldsymbol{\theta}$ to be the realization parameters of the source language, as estimated from the source treebank $\boldsymbol{y}$. This is at least a linguistically realistic $\boldsymbol{\theta}$, although it may

[8]We also used the minibatch to estimate the average sentence length $\mathbb{E}_{\mathbf{s}\sim p}[\,|\mathbf{s}|\,]$ in Equation (6.4), although here we could have simply used all of $\boldsymbol{y}$ since this value does not change.

not be close to the target language.[9]

For this initial estimation, we follow Chapter 3 and perform supervised training on $\mathcal{Y}$ of the log-linear realization model Equation (6.1), by maximizing the conditional log-likelihood of $\mathcal{Y}$, namely $\sum_{(\mathbf{t},\mathbf{o})\in\mathcal{Y}} \log p_\theta(\mathbf{o} \mid \mathbf{t})$, where $(\mathbf{t}, \mathbf{o})$ are an unordered tree and its observed ordering in $\mathcal{Y}$. This initial objective is convex.[10]

6.5 Experiments

6.5.1 Data and setup

Again, we use Universal Dependencies version 1.2 (Nivre et al., 2015)—a set of 37 dependency treebanks for 33 languages, with a unified POS-tag set and relation label set.

Our evaluation metric was unlabeled attachment score (UAS) when parsing a target treebank with a parser trained on a (possibly permuted) source treebank. For both evaluation and training, we used only the training portion of each treebank.

Following Chapter 3, we use the Yara parser (Rasooli and Tetreault, 2015) with the same modification as mentioned in Section 3.6 to ignore the input words and use only the input *gold* POS tags (see Section 6.1.3). To train the Yara parser on a (possibly permuted) source treebank, we first train on 80% of the trees and use the remaining 20% to tune Yara's hyperparameters. We then retrain Yara on 100% of the source trees and evaluate it on the target treebank.

[9]As an improvement, one could also try initial realization parameters for $\mathcal{Y}$ that are estimated from treebanks of *other* languages. Concretely, the optimizer could start by selecting a "galactic" treebank from Chapter 3 that is already close to the target language, according to Equation (6.4), and try to make it even closer. We leave this to future work.

[10]Unfortunately, we did not regularize it, which probably resulted in initializing some parameters too close to $\pm\infty$ for the optimizer to change them meaningfully.

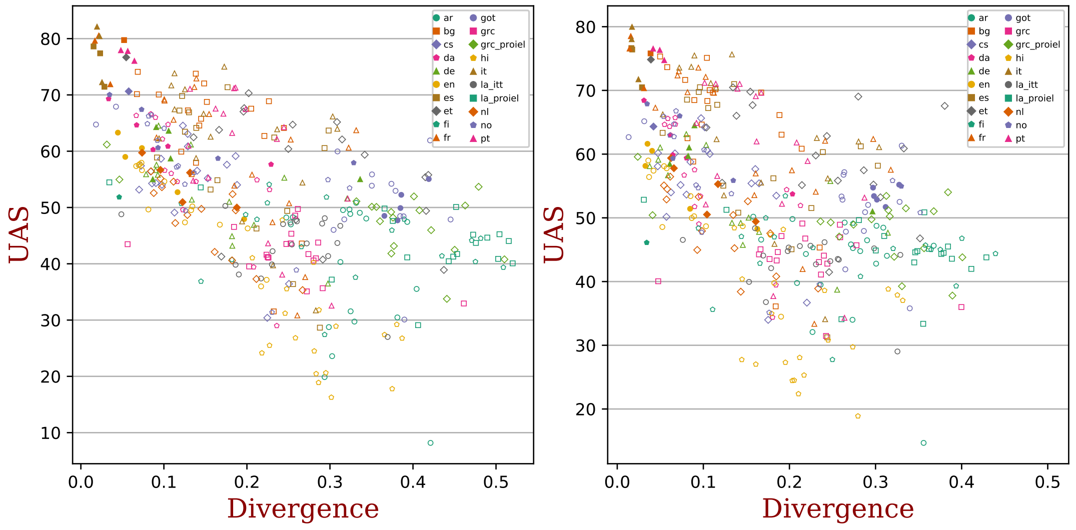


Figure 6.1: UAS is higher when divergence is lower. Each point represents a pair of source and target languages, whose shape and color identify the treebank of the target language (see legend). The marker is solid if the source and target languages belong to the same language family.[13]The left graph uses the original source treebank (Kendall's $\tau = -0.41$), while the right graph uses its permuted version ($\tau = -0.39$).

We adopt the same set up as Section 4.6, which uses 20 treebanks (18 distinct languages) in Table 4.2 as *development data*, and hold out the remaining 17 treebanks for the final evaluation. We perform a grid search that evaluated all $(\alpha_1, \alpha_2, \beta)$ triples of Equation (6.4) in $\{0.0, 0.2, \ldots, 1\}^3$ and chose $(\alpha_1, \alpha_2, \beta) = (0.2, 1, 0.2)$, which maximizes the target-language UAS, averaged over all 376 transfer experiments where the source and target treebanks were development treebanks of different languages.[11]

The next few sections perform some exploratory analysis on these 376 experiments. Then, for the final test in Section 6.5.5, we will evaluate UAS on all 337 transfer experiments where the source is a development treebank and the target is a test treebank of a different language.[12]

[11]We have 19*20=380 pairs in total, minus the four excluded pairs (grc, grc_proiel), (grc_proiel, grc), (la_proiel, la_itt) and (la_itt, la_proiel).

[12]Specifically, there are 3 duplicated sets: {grc, grc_proiel}, {la, la_proiel, la_itt}, and {fi, fi_ftb}. Whenever one treebank is used as the target language, we exclude the other treebanks in the same set.

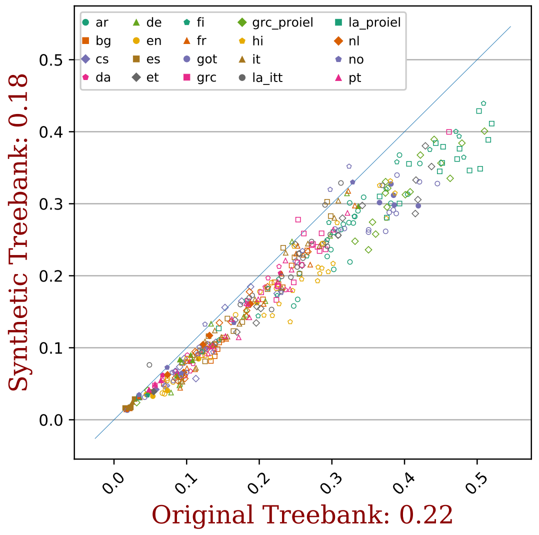


Figure 6.2: This graph plots the x-axes from the two graphs in Figure 6.1 against each other. We see that for almost every source-target pair (330/376 = 96.01% of the pairs), the SGD optimizer succeeded in constructing a permuted source treebank $\boldsymbol{y}'$ with lower divergence to the target than the original source treebank $\boldsymbol{y}$. The diagonal line $y = x$ is also shown for readability. The number on each axis is the mean value.

6.5.2 Exploratory analysis

We have assumed that a smaller divergence between source and target treebanks results in better transfer parsing accuracy. Figure 6.1 shows that these quantities are indeed correlated, both for the original source treebanks and for their "made to order" permuted versions.

Thus, we hope that the optimizer will find a systematic permutation that reduces the divergence. Does it? Yes: Figures 6.2 and 6.3 show that the optimizer almost always manages to reduce the objective on training data, as expected.

One concern is that our divergence metric might misguide us into producing dysfunctional languages whose trees cannot be easily recovered from their surface strings, i.e., they have no good parser. In such a language, the word order might

[13] According to the (sub-)family information in Table 3.3.

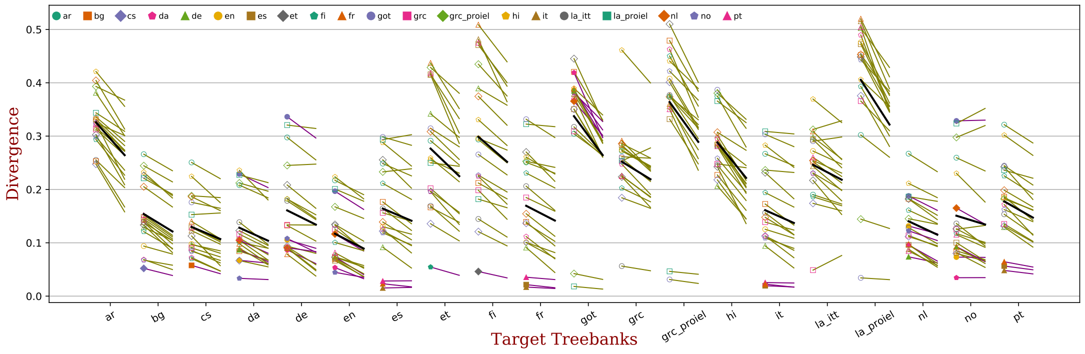


Figure 6.3: Divergences between 376 pairs of development treebanks. This is a different presentation of Figure 6.2 in which the source-target pairs are grouped into columns. Each column represents a target treebank, and each line segment within that column shows the divergence Equation (6.4) from variants of a different source treebank. The two points on that segment (from left to right) represent the original source treebank and its "made to order" permutation. We use solid markers and purple lines if the transfer is *within-family* (source and target treebank from the same language family), and hollow and olive for *cross-family* transfer. The **black** segment in each column is the mean of the others.

be extremely free (e.g., $\boldsymbol{\theta} = \mathbf{0}$), or common constructions might be syntactically ambiguous. Fortunately, Section 6.5.3 shows that our synthetic languages appear natural with respect to their their parsability.

The above findings are promising. So does permuting the source language in fact result in better transfer parsing of the target language? We experiment on the 376 development pairs.

The solid lines in Figure 6.5 show our improvements on the dev data, with a simpler scatterplot given by in Figure 6.6. The upshot is that the synthetic source treebanks yield a transfer UAS of 52.92 on average. This is not yet a result on held-out test data: recall that 52.92 was the best transfer UAS achieved by any hyperparameter setting. That said, it is 1.00 points better than transferring from the original source treebanks, a significant difference (paired permutation test by language pair, $p < 0.01$).

Figure 6.5 shows that this average improvement is mainly due to the many cases where the source and target languages come from different families. Permutation tends to improve source languages that were doing badly to start with. However, it tends to hurt a source language that is already in the target language family.

A hypothetical experiment shows that permuting the source does have good *potential* to help (or at least not hurt) in both cases. The dashed lines in Figure 6.5—and the scatterplot in Figure 6.7—show the potential of the method, by showing the improvement we would get from permuting each source treebank using an "oracle" realization policy—the supervised realization parameters $\boldsymbol{\theta}$ that are estimated from the actual target treebank. The usefulness of this oracle-permuted source varies depending on the source language, but it is usually much better than the automatically-permuted version of the same source.

This shows that large improvements would be possible if we could only find the best permutation policy allowed by our model family. The question for future work is whether such gains can be achieved by a more sensitive permutation model than Equation (6.1), a better divergence objective than Equation (6.4), or a better search algorithm than Section 6.4.2. Identifying the best available source treebank, or the best mixture of all source treebanks, would also help greatly.

6.5.3 Parsability

For reasons explained above, we evaluated the parsability of our "made to order" synthetic languages as shown in Figure 6.4, when the parser was given only POS sequences as input. For each synthetic treebank $\boldsymbol{y}'$, we trained the Yara parser on a training portion and evaluated its UAS on a development portion. In fact, the synthetic treebanks were slightly *more* parsable than the originals (mean UAS of

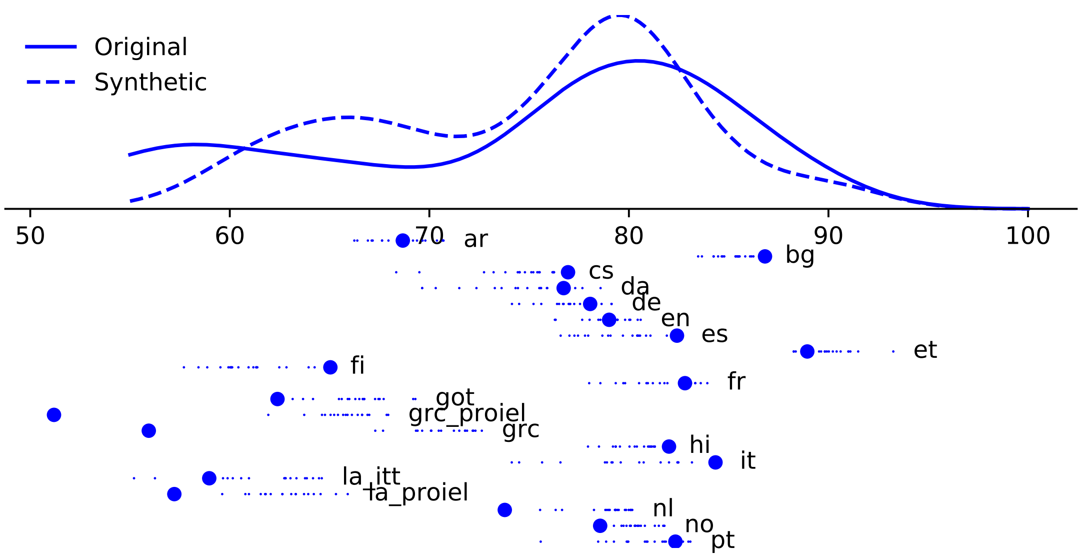


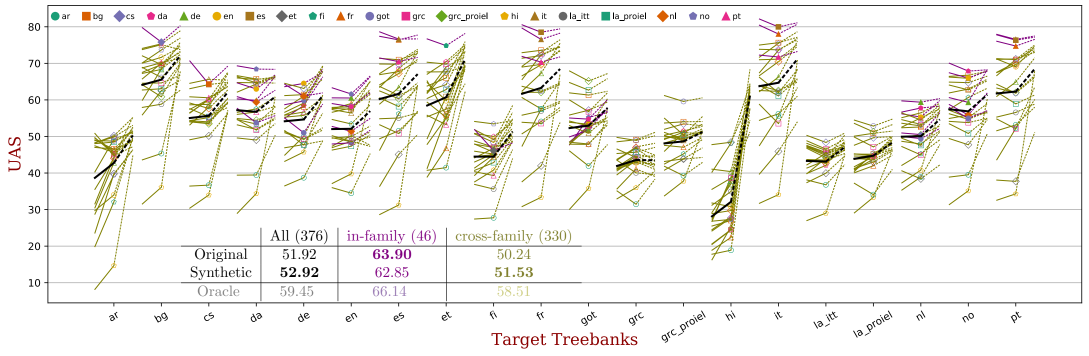
Figure 6.4: Parsability of 20 real treebanks vs. their many synthetic re-realizations (cf. Figure 3.2).

74.96 vs. 73.61), though the improvement was far from significant under an unpaired permutation test ($p = 0.48$). By contrast, Chapter 3 produces synthetic treebanks that were significantly *less* parsable. We observed some regression to the mean: highly parsable treebanks usually became less parsable when permuted, and vice-versa.

6.5.4 Sensitivity to initializer

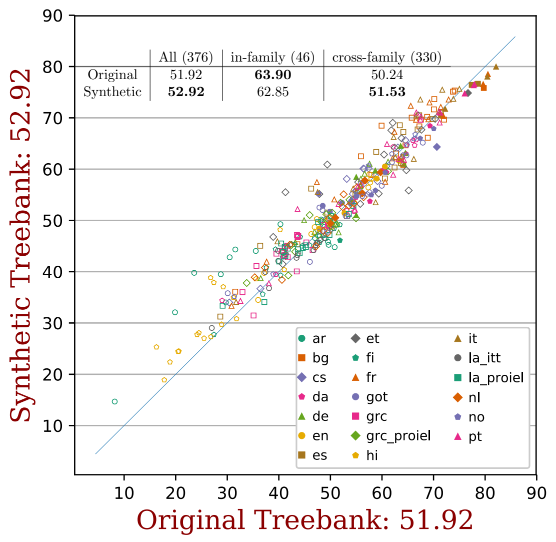
Figure 6.5 makes clear that performance of the synthetic source treebanks is strongly correlated with that of their original versions. Most points in Figure 6.6 lie near the diagonal (Kendall's $\tau = 0.85$). Even with oracle permutation in Figure 6.7, the correlation remains strong ($\tau = 0.59$), suggesting that the choice of source treebank is important even beyond its effect on search initialization.

[14]For speed, we restricted the experiment of Figure 6.8 to choose 48 of the 376 pairs. The source treebanks were en, no, de, es, fr, pt, hi, it, ar. The target treebanks were fr, hi, de, ar, pt, en. This covers both in-family transfer and cross-family transfer. By excluding the cases where source = target, we got $9 * 6 - 6 = 48$ pairs.



	All (376)	in-family (46)	cross-family (330)
Original	51.92	**63.90**	50.24
Synthetic	**52.92**	62.85	**51.53**
Oracle	59.45	66.14	58.51

Figure 6.5: UAS from 376 pairs of development treebanks. Each column represents a target treebank, and each polyline within that column shows transfer from variants of a different source treebank. The three points on the polyline (from left to right) represent the target UAS for parsers trained on three sources: the original source treebank, the "made to order" permutation that attempts to match surface statistics of the target treebank, and an oracle permutation that uses a realization model trained on the target language. We use solid markers and purple lines if the transfer is *within-family* (source and target treebank from the same language family), and hollow and olive for cross-family transfer. The **black** polyline in each column is the mean of the others. The table in the lower left gives summary results; the number in each column header gives the number of points summarized. For each column, we boldface the better result between the "Synthetic" and "Original", or both if they are not significantly different (paired permutation test, $p < 0.01$). We also show the oracle permutation result in row "Oracle".



	All (376)	in-family (46)	cross-family (330)
Original	51.92	**63.90**	50.24
Synthetic	**52.92**	62.85	**51.53**

Figure 6.6: Unlabeled attachment scores (UAS) on 376 treebank pairs within the development languages. Each marker represents one pair, whose x-axis is the UAS on the target language using the original treebank of the source language, and the y-axis is the UAS using the synthetic treebank permuted from the original treebank. The table in the upper left gives summary results; the number in each column header gives the number of points summarized. For each column, we boldface the better result, as well as the other if it is not significantly worse (paired permutation test, $p < 0.01$).

We suspected that when “made to order” source treebanks (more than the oracle versions) have performance close to their original versions, this is in part because the optimizer can get stuck near the initializer (Section 6.4.3). To examine this, we experimented with random restarts, as follows. In addition to informed initialization (Section 6.4.3), we optimized from 5 other starting points $\boldsymbol{\theta} \sim \mathcal{N}(\mathbf{0}, I)$. From these 6 runs, we selected the final parameters that achieved the best divergence Equation (6.4). As shown by Figure 6.8 in the supplement, greater gains appear to be possible with more aggressive search methods of this sort, which we leave to future work. We could also try non-random restarts based on the realization parameters of other languages, as suggested in Footnote 9.

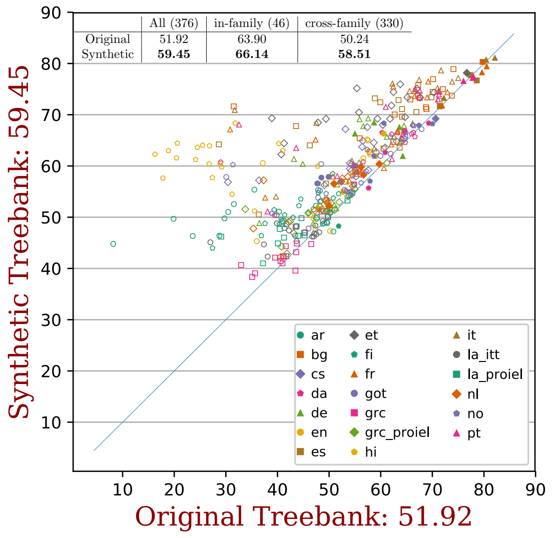


Figure 6.7: UAS on 376 language pairs within the training languages. The design is similar to Figure 6.6, but the synthetic treebanks are generated using an oracle—the actual realization model of the target language.

6.5.5 Final evaluation on the test languages

For our final evaluation, we use the same hyperparameters (Section 6.5.1) and report on single-source transfer to the 17 held-out treebanks.

The development results hold up in Figure 6.9. Using the synthetic languages yields 50.36 UAS on average—1.75 points over the baseline, which is significant (paired permutation test, $p < 0.01$).

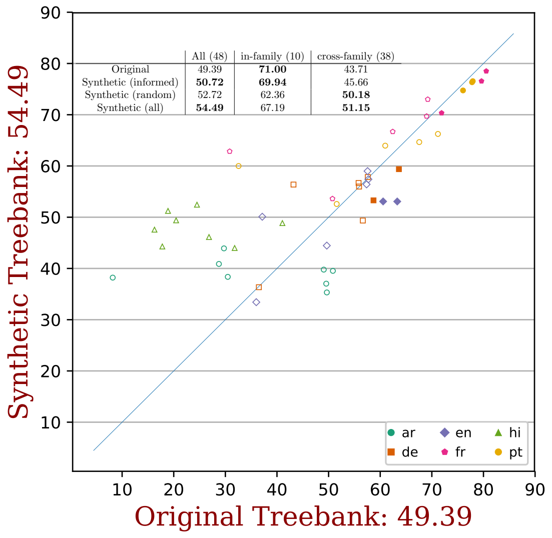
Table 6.1 gives a breakdown view on each language pair for the above development and test results.

6.6 Related Work

Our novel proposal ties into the recent interest in data augmentation in supervised machine learning. In unsupervised parsing, the most widely adopted synthetic data

	bg	es	grc_proiel	ar	en	la_proiel	la_itt	fi	de	fr	it	got	pt	no	et	nl	hi	cs	grc	da
bg	-	69.66	**60.85**	**45.34**	**71.65**	**63.05**	**58.83**	**68.48**	**68.34**	**70.04**	**75.11**	66.13	**70.18**	**73.65**	**62.50**	**69.67**	**36.11**	75.81	**64.64**	**75.33**
es	70.99	-	60.32	**51.54**	**67.74**	58.18	**55.05**	**56.21**	**63.34**	76.42	76.64	61.23	70.49	**70.50**	**45.07**	**67.23**	**31.25**	**69.76**	50.81	**68.55**
grc_proiel	**54.02**	**49.28**	-	39.27	**50.23**	50.42	**43.89**	**45.23**	**49.77**	**47.06**	48.93	59.58	49.44	**51.04**	**43.81**	**51.20**	**37.80**	**53.44**	-	**51.50**
ar	46.58	44.78	45.63	-	**34.00**	**48.46**	**49.82**	**32.08**	**42.81**	46.48	45.83	**48.75**	45.25	**39.50**	**39.78**	**44.04**	**14.68**	**50.18**	**49.26**	**44.33**
en	**57.78**	**57.40**	**48.69**	34.49	-	**47.34**	**49.97**	**53.42**	**60.52**	**59.00**	**56.41**	**48.26**	**48.56**	61.62	48.68	**51.42**	**39.77**	**58.11**	**50.25**	58.15
la_proiel	**50.87**	45.14	**51.26**	**34.09**	**44.34**	-	-	**44.88**	**43.80**	**41.99**	**43.58**	**52.84**	44.78	**45.50**	**43.01**	**44.51**	**33.37**	**49.65**	47.15	44.59
la_itt	45.57	**46.18**	44.19	36.78	43.20	-	-	**44.08**	**43.44**	43.55	44.78	45.21	**45.62**	**45.34**	**39.95**	**42.71**	**29.03**	48.37	46.54	**42.10**
fi	47.00	**46.78**	**45.02**	**27.75**	49.15	**42.86**	35.62	-	**45.70**	**44.38**	**45.01**	**45.32**	39.30	53.44	46.12	**45.18**	**40.81**	48.38	47.07	**49.99**
de	**61.44**	**61.05**	**55.77**	**38.72**	**64.51**	47.66	49.20	**50.03**	-	**58.11**	**59.12**	51.00	**56.68**	**59.71**	**47.79**	61.03	**45.75**	63.13	49.22	**58.45**
fr	**73.57**	78.51	**62.09**	**54.09**	**69.71**	57.54	**56.97**	**57.46**	**67.28**	-	76.56	**62.37**	70.34	**73.00**	**41.96**	**69.62**	**33.36**	**72.12**	**53.56**	**72.35**
it	**75.65**	79.97	62.53	**56.19**	**71.14**	61.09	**62.34**	**55.53**	**66.24**	78.03	-	61.98	**71.74**	**75.48**	**45.91**	**70.45**	**34.09**	**73.70**	53.53	**73.57**
got	**61.33**	**53.35**	65.16	41.92	**53.42**	62.67	47.83	**52.03**	**51.71**	47.94	**50.89**	-	**52.85**	**55.20**	**52.51**	**52.85**	**35.80**	**57.17**	56.76	**54.74**
pt	71.02	76.34	61.99	**53.17**	69.09	58.92	**56.57**	**52.20**	**64.89**	74.74	76.55	61.82	-	**70.26**	37.72	**69.62**	**34.31**	**71.19**	52.10	**71.04**
no	**66.77**	62.74	**55.85**	**39.53**	65.99	50.82	**54.71**	**60.67**	59.33	**62.97**	**65.91**	54.97	55.14	-	47.73	55.86	**35.14**	64.72	53.79	67.88
et	**66.02**	**60.89**	**67.57**	**41.48**	59.79	**62.84**	**55.50**	74.84	55.22	**46.78**	**57.47**	**69.03**	53.22	**67.69**	-	55.84	**55.14**	**64.18**	69.80	**70.47**
nl	**52.60**	**56.46**	**50.44**	**38.91**	55.29	47.57	47.93	**45.24**	59.38	**52.89**	**55.09**	**49.42**	**54.53**	50.52	38.41	-	**40.81**	**53.30**	**44.96**	**57.79**
hi	**27.02**	**24.45**	**37.04**	**18.89**	30.81	**37.88**	**34.96**	**48.18**	**40.39**	**22.38**	**25.31**	**38.82**	**28.07**	**27.31**	**48.42**	**29.74**	-	**27.74**	**38.60**	**24.50**
cs	64.33	**64.21**	53.48	**36.69**	**53.65**	**55.41**	**54.00**	**58.09**	**58.78**	**60.03**	**65.64**	55.42	**60.58**	**60.11**	**50.16**	**57.66**	**33.99**	-	**55.16**	**60.16**
grc	**49.11**	**43.06**	-	31.46	**42.81**	**45.70**	40.05	**43.53**	**44.07**	**41.10**	**43.31**	48.92	**44.63**	**46.76**	**45.07**	**46.96**	**36.00**	**44.27**	-	**47.14**
da	**65.72**	**64.69**	**54.42**	**39.46**	62.99	**51.93**	**53.39**	**57.54**	59.87	**64.33**	**65.58**	53.74	**56.70**	68.43	**49.03**	**59.39**	**34.39**	64.65	51.76	-
cu	**64.84**	**55.15**	64.42	45.55	**56.87**	65.82	**49.97**	**54.62**	**52.16**	**50.95**	**54.11**	68.34	**55.78**	**59.22**	**54.91**	**54.12**	**33.59**	**60.19**	60.35	**58.69**
el	62.69	56.82	58.68	**45.80**	**59.49**	50.34	**57.11**	48.52	61.31	58.95	57.99	57.61	59.92	**60.82**	41.97	56.08	**39.93**	64.70	**58.24**	**57.97**
eu	**48.68**	**40.02**	**45.31**	**32.79**	**44.84**	**46.14**	**43.73**	**41.29**	43.24	**34.01**	**44.53**	**42.07**	**43.28**	**43.29**	**48.12**	**47.11**	**48.03**	**47.60**	43.22	**46.83**
fa	**50.33**	**48.78**	42.43	**45.96**	**38.08**	**48.37**	**49.03**	**39.40**	**45.34**	**48.73**	**48.11**	**50.91**	**47.47**	**40.97**	**38.37**	**43.30**	**28.86**	**54.08**	**49.82**	**44.22**
fi_ftb	**49.76**	**47.07**	**51.10**	**31.12**	**50.56**	**46.65**	37.79	-	**51.57**	**42.72**	**49.08**	**48.42**	**47.99**	**54.20**	48.06	**48.54**	**46.93**	**50.03**	48.29	46.35
ga	55.21	52.82	**53.29**	**55.77**	51.58	49.57	**51.01**	**46.43**	**52.90**	**55.08**	55.33	53.00	53.96	**56.35**	**43.84**	**50.51**	**29.18**	**61.18**	**50.65**	**58.77**
he	59.80	57.91	61.15	**55.17**	**52.93**	53.43	**56.49**	**50.67**	**53.00**	52.12	56.90	61.75	57.22	**56.07**	**42.57**	**53.60**	**28.62**	62.36	**55.46**	**53.86**
hr	64.23	62.44	**52.19**	**37.70**	**55.99**	**55.34**	**54.35**	**58.48**	**57.13**	**57.74**	**64.86**	**55.13**	57.31	**53.39**	**48.44**	55.28	**33.35**	**69.26**	**48.08**	**60.81**
hu	**56.65**	**50.57**	**53.06**	**28.23**	**54.81**	**47.40**	43.08	**53.83**	**57.33**	**49.67**	**50.87**	**48.79**	**51.16**	**56.98**	50.03	56.10	**53.50**	53.15	**54.38**	**54.74**
id	**62.73**	58.01	49.84	**48.08**	**37.91**	52.21	**43.00**	**49.40**	**41.87**	53.28	62.00	52.95	56.84	**50.61**	**36.96**	**47.48**	**22.80**	**62.00**	**44.87**	**53.85**
ja_ktc	**20.87**	**18.85**	**35.94**	**14.36**	**28.50**	**37.55**	**28.39**	**50.45**	31.34	**17.89**	**17.86**	**30.82**	**20.52**	**29.15**	**44.33**	16.67	62.09	**25.05**	**37.95**	**28.65**
la	**46.83**	**39.91**	**43.68**	**30.04**	**40.52**	-	-	**43.91**	**38.81**	**36.32**	**38.62**	**46.01**	**41.86**	**42.55**	**42.06**	**40.97**	**35.62**	43.28	45.64	**43.49**
pl	65.75	**64.74**	53.20	53.27	**57.12**	57.79	58.31	**57.59**	**58.78**	**60.78**	**64.39**	54.91	63.64	**61.70**	**60.45**	**56.73**	**37.49**	69.97	**64.60**	**63.49**
ro	**67.44**	64.81	**55.10**	**51.82**	**59.18**	**53.32**	**56.65**	**54.98**	**57.10**	62.13	65.79	54.93	57.00	**60.97**	**46.47**	**55.91**	**28.52**	**65.48**	**55.33**	**64.64**
sl	70.37	**69.47**	**56.33**	**39.64**	**63.15**	**56.11**	57.49	**63.81**	**67.06**	**68.35**	**69.23**	57.37	**57.92**	**66.13**	**54.58**	**60.26**	**38.36**	76.41	**60.16**	**66.34**
sv	**68.72**	**66.99**	**58.19**	**39.60**	**69.88**	**55.46**	**57.12**	**64.25**	**65.37**	**65.67**	**69.35**	58.62	**61.06**	**75.33**	53.59	**64.17**	**39.60**	**68.41**	**59.44**	73.22
ta	**40.48**	**27.35**	**39.34**	**17.37**	**42.11**	**40.89**	**37.11**	**42.32**	39.48	**28.80**	**30.86**	**32.91**	**30.47**	**39.92**	**44.93**	31.90	**57.59**	**33.10**	33.79	**31.21**

Table 6.1: UAS scores on single-source transfer results using the synthetic languages, where the columns represent source treebanks and the rows represent target treebanks. The upper half of the table is the cross-validation result used for generating the y-axis of Figure 6.6. The lower half is the final test result used for the y-axis of Figure 6.9. For each pair, we boldface the results that are not significantly worse (paired permutation test by sentence, $p < 0.05$) than using the original treebanks.



	All (48)	in-family (10)	cross-family (38)
Original	49.39	**71.00**	43.71
Synthetic (informed)	**50.72**	**69.94**	45.66
Synthetic (random)	52.72	62.36	**50.18**
Synthetic (all)	**54.49**	67.19	**51.15**

Figure 6.8: UAS on 48 of the language pairs within the development languages.[14] The design is similar to Figure 6.6, but we optimize divergence more aggressively by selecting the best of 6 optimization runs for each pair (informed initialization plus 5 random restarts). In 36 of 48 cases, the best run used a random restart. The average x and y values are given in the first and last rows of the table, with the intermediate rows showing the results if we had used *only* informed initialization or *only* random restarts. Each column boldfaces the best result as well as all others that are not significantly worse (paired permutation test, $p < 0.01$).

method has been *annotation projection*, which generates synthetic analyses of target-language sentences by "projecting" the analysis from a source-language translation. Of course, this requires bilingual corpora as an additional resource. Annotation projection was proposed by Yarowsky, Ngai, and Wicentowski (2001), gained promising results on sequence labelling tasks, and was later developed for unsupervised parsing (Hwa et al., 2005; Ganchev, Gillenwater, and Taskar, 2009; Smith and Eisner, 2009; Tiedemann, 2014; Ma and Xia, 2014; Tiedemann, Agić, and Nivre, 2014). Recent work in this vein has mainly focused on improving the synthetic data, including reweighting the training trees (Agić et al., 2016) or pruning those that cannot be aligned well (Rasooli and Collins, 2015; Rasooli and Collins, 2017; Lacroix et al., 2016).

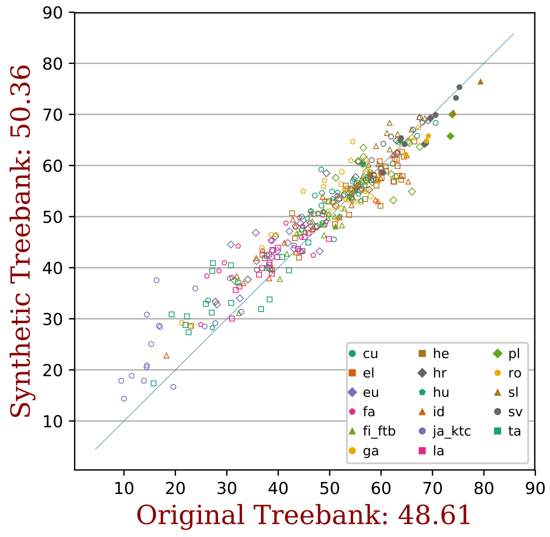


Figure 6.9: UAS on 337 treebank pairs from the developments languages to the test languages.

On the other hand, Chapter 3 proposed to permute source language treebanks using word order realization models trained on other source languages. They generated on the order of 50,000 synthetic languages by "mixing and matching" a few dozen source languages. Their idea was that with a large set of synthetic languages, they could use them as supervised examples to train an unsupervised structure discovery system that could analyze any new language. Systems built with this dataset were competitive in single-source parser transfer (Chapter 3), typology prediction (Chapter 4), and parsing unknown languages (Chapter 5).

This chapter differs from Chapter 3 in that our synthetic treebanks are "made to order." Rather than combine aspects of different treebanks and hope to get at least one combination that is close to the target language, we "combine" the source treebank with a POS corpus of the target language, which guides our customized permutation of the source.

Beyond unsupervised parsing, synthetic data has been used for several other tasks.

In NLP, it has been used for complex tasks such as question answering (QA) (Serban et al., 2016) and machine reading comprehension (Weston et al., 2016; Hermann et al., 2015; Rajpurkar et al., 2016), where highly expressive neural models are used and not enough real data is available to train them. In the playground of supervised parsing, Gulordava and Merlo (2016) conduct a controlled study on the parsibility of languages by generating treebanks with short dependency length and low variability of word order.

6.7 Conclusion and Future Work

We have shown in this chapter on how to permute the source treebank to better resemble the target language on the surface (in its distribution of gold POS bigrams), which could improve cross-lingual transfer parsing. The code is available at `https://github.com/wddabc/ordersynthetic`. The key idea is grounded in the notion that by trying to explain the POS bigram counts in a target corpus, we can discover a stochastic realization policy for the target language, which correctly "translates" the source trees into appropriate target trees.

We formulated an objective for evaluating such a policy, based on KL-divergence between bigram models. We showed that the objective could be computed efficiently by dynamic programming, thanks to the limitation to bigram statistics.

Experimenting on the Universal Dependencies treebanks v1.2, we showed that the synthetic treebanks were—on average—modestly but significantly better than the corresponding real treebanks for single-source transfer.

On the downside, Figure 6.6 shows that with our current method, permuting the source language to be more like the target language is helpful (on average) only when the source language is from a different language family. This contrast would be even more striking if we had a better optimizer: Figure 6.8 shows that SGD's initialization bias limits permutation's benefit for cross-family training, as well as its harm for within-family training.

Several opportunities for future work have already been mentioned throughout this chapter (Sections 6.2.1, 6.5.2 and 6.5.4 and Footnote 9). First of all, since this chapter uses the same permutation distribution family as Chapter 3, some improvements as discussed in Section 3.8 will also be beneficial to this chapter, e.g. producing non-projective trees. In addition, we are interested in experimenting with richer families of permutation distributions, as well as "conservative" distributions that tend to prefer the original source order. We could use entropy regularization (Grandvalet and Bengio, 2005) to encourage more "deterministic" patterns of realization in the synthetic languages.

It is possible to combine the approaches in this chapter with Chapter 3. For example, the on-demand permutation in this chapter only finds a local optimum of the realization model, so initialization is important. The mix-and-match permutations of the real training language in Chapter 3 could be used as different plausible initializations. Finally, train a universal parser on these permuted models. This is a form of *local* learning but with synthetic training data created in the vicinity of the test point.

We would also like to consider more sensitive divergence measures that go beyond bigrams, for example using recurrent neural network language models (RNNLMs) for $\hat{q}$ and $\hat{p}_{\boldsymbol{\theta}}$. This means abandoning our exact dynamic programming methods; we would also like to abandon exact exhaustive enumeration in order to drop Section 6.4.1's bounds on n. Fortunately, there exist powerful MCMC methods (Eisner and Tromble,

2006) that can sample from interesting distributions over the space of $n!$ permutations, even for large n. Thus, we could approximately sample from $p_{\boldsymbol{\theta}}$ by drawing permuted versions of each tree in $\boldsymbol{y}$.

Given this change, a very interesting direction would be to graduate from POS language models to word language models, using cross-lingual unsupervised word embeddings (Ruder, Vulić, and Søgaard, 2017). This would eliminate the need for the gold POS tags that we unrealistically assumed (which are typically unavailable for a low-resource target language). Furthermore, it would enable us to harness richer lexical information beyond the 17 UD POS tags. After all, even a (gold) POS corpus might not be sufficient to determine the word order of the target language: "`NOUN VERB NOUN`" could be either subject-verb-object or object-verb-subject. However, "`water drink boy`" is presumably object-verb-subject. Thus, using cross-lingual embeddings, we would try to realize the unordered source trees so that their word strings, with few edits, can achieve high probability under a neural language model of the target.

Chapter 7

Conclusion

We have presented the amortized Bayes (AB) framework to estimate the parsing parameters for unsupervised dependency parsing. The main novelty of our approach is converting this traditional unsupervised learning problem into a supervised one—we train our system on many synthetic languages. Around this idea, we introduced four published papers:

- **The Galactic Dependencies** (Chapter 3; Wang and Eisner, 2016), which aims to solve the challenge of data sparsity in our approach (Sections 1.4 and 2.5) by introducing a large set of synthetic languages generated from the mix-and-match over the real languages. This resource serves as our main working data throughout this thesis.

- **Fine-grained prediction of syntactic typology** (Chapter 4; Wang and Eisner, 2017), which shows that the AB estimator trained on our synthetic data is *effective* in predicting syntactic typology from only POS corpus—another unsupervised learning problem in natural language processing (NLP). In addition to the importance on its own, the unsupervised prediction of syntactic typology could be considered preliminary task of unsupervised parsing.

- **Unsupervised dependency parsing** (Chapter 5; Wang and Eisner, 2018a), which shows that the AB estimator *outperforms* various baselines on unsupervised parsing—our main task. We conclude that the source of the improvement is two fold: 1) Training on thousands of synthetic languages, and 2) surface-form features extracted from the unparsed corpus of the target language.

- **Synthetic data made to order** (Chapter 6; Wang and Eisner, 2018b), which extends the idea of GD and proposes to generate synthetic data whose surface statistics match the target language. Experimental results on single-source transfer parsing show improvements over using only real languages, especially when source and target languages are from different families.

Throughout this thesis, we are trying to answer a fundamental question on whether the surface statistics of a language provide clues about how to find the underlying syntactic dependencies. Chomsky (1965) imagined that such clues might be exploited by a Language Acquisition Device, so it is interesting to know that they do *exist*, at least in the surface part-of-speech corpus.

Another takeaway message is that synthetic training languages are useful for NLP. Using synthetic examples in training is a way to encourage a system to be invariant to superficial variation. We created synthetic languages by varying the surface structure in a way that "should" preserve the deep structure. This allows our trained system to be invariant to variation in surface structure, just as object recognition wants to be invariant to an image's angle or lighting conditions (Chapters 3 and 6).

Our final takeaway goes beyond language: one can treat unsupervised structure discovery as a supervised learning problem. As Sections 1.4 and 2.5 discuss, this approach inherits the advantages of supervised learning. Training may face an easier optimization landscape, and we can train the system to find the *specific* kind of structure that we desire, using any features that we think may be discriminative.

Future work has been discussed in Sections 3.8, 4.8, 5.7 and 6.7. In sum, the general direction is two-fold: 1) Generate better synthetic languages by considering a wider range of linguistic complexities, where possible approaches include richer features in the reordering model, handling non-projective trees, and generating synthetic words; and 2) improve the parsing architecture, where possible approaches include using more recent local parsers such as Dozat and Manning (2017), using attention mechanisms to extract surface-form features, and relaxing the delexicalized assumption on the target languages by using cross-language word representation.

References

Abadi, Martín, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. URL: https://www.tensorflow.org/.

Abu-Mostafa, Yaser S. (1995). "Hints". In: *Neural Computation* 7, pp. 639–671. URL: http://www.work.caltech.edu/yaser/paper/95hints.ps.

Agarwal, Apoorv, Boyi Xie, Ilia Vovsha, Owen Rambow, and Rebecca Passonneau (2011). "Sentiment Analysis of Twitter Data". In: *Proceedings of the Workshop on Language in Social Media (LSM 2011)*. Portland, Oregon: Association for Computational Linguistics, pp. 30–38. URL: https://www.aclweb.org/anthology/W11-0705.

Agić, Željko, Anders Johannsen, Barbara Plank, Héctor Martínez Alonso, Natalie Schluter, and Anders Søgaard (2016). "Multilingual Projection for Parsing Truly Low-Resource Languages". In: *Transactions of the Association for Computational Linguistics* 4, pp. 301–312. URL: http://aclweb.org/anthology/Q16-1022.

Ammar, Waleed, George Mulcaire, Miguel Ballesteros, Chris Dyer, and Noah A. Smith (2016). "Many Languages, One Parser". In: *Transactions of the Association for Computational Linguistics* 4, pp. 431–444. DOI: 10.1162/tacl_a_00109. URL: https://www.aclweb.org/anthology/Q16-1031.

Andor, Daniel, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins (2016). "Globally Normalized

Transition-Based Neural Networks”. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, pp. 2442–2452. DOI: 10.18653/v1/P16-1231. URL: https://www.aclweb.org/anthology/P16-1231.

Angluin, Dana and Carl H Smith (1983). “Inductive inference: Theory and methods”. In: *ACM Computing Surveys (CSUR)* 15.3, pp. 237–269.

Arjovsky, Martin, Soumith Chintala, and Léon Bottou (2017). “Wasserstein Generative Adversarial Networks”. In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. International Convention Centre, Sydney, Australia: PMLR, pp. 214–223. URL: http://proceedings.mlr.press/v70/arjovsky17a.html.

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2015). “Neural Machine Translation by Jointly Learning to Align and Translate”. In: *Proceedings of the International Conference on Learning Representations*.

Baker, J. K. (1979). “Trainable Grammars for Speech Recognition”. In: *Speech Communication Papers Presented at the 97th Meeting of the Acoustical Society of America*. Ed. by Jared J. Wolf and Dennis H. Klatt. MIT, Cambridge, MA.

Basu, Sumit, Chuck Jacobs, and Lucy Vanderwende (2013). “Powergrading: A Clustering Approach to Amplify Human Effort for Short Answer Grading”. In: *Transactions of the Association for Computational Linguistics* 1, pp. 391–402. URL: https://transacl.org/ojs/index.php/tacl/article/view/139.

Bender, Emily M. (2009). “Linguistically Naïve != Language Independent: Why NLP Needs Linguistic Typology”. In: *Proceedings of the EACL 2009 Workshop on the Interaction between Linguistics and Computational Linguistics: Virtuous, Vicious or Vacuous?*, pp. 26–32. URL: http://www.aclweb.org/anthology/W09-0106.

Blunsom, Phil and Trevor Cohn (2010). “Unsupervised induction of tree substitution grammars for dependency parsing”. In: *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. Cambridge, MA: Association for Computational Linguistics, pp. 1204–1213.

Borg, Ingwer and Patrick J.F. Groenen (2005). *Modern Multidimensional Scaling: Theory and Applications*. URL: http://www.springer.com/us/book/9780387251509.

Bowman, Samuel R., Christopher Potts, and Christopher D. Manning (2015). "Recursive Neural Networks Can Learn Logical Semantics". In: *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality*. Beijing, China: Association for Computational Linguistics, pp. 12–21. DOI: 10.18653/v1/W15-4002. URL: https://www.aclweb.org/anthology/W15-4002.

Brin, Sergey and Lawrence Page (1998). "The anatomy of a large-scale hypertextual web search engine". In: *Computer networks and ISDN systems* 30.1-7, pp. 107–117.

Carroll, Glenn and Eugene Charniak (1992). "Two Experiments on Learning Probabilistic Dependency Grammars from Corpora". In: *Working Notes of the AAAI Workshop on Statistically-Based NLP Techniques*. Department of Computer Science, Univ., pp. 1–13. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.52.9158.

Chen, Danqi and Christopher Manning (2014). "A Fast and Accurate Dependency Parser using Neural Networks". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, pp. 740–750. DOI: 10.3115/v1/D14-1082. URL: https://www.aclweb.org/anthology/D14-1082.

Chiang, David (2005). "A Hierarchical Phrase-Based Model for Statistical Machine Translation". In: *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*. Association for Computational Linguistics. Ann Arbor, Michigan: Association for Computational Linguistics, pp. 263–270. DOI: 10.3115/1219840.1219873. URL: https://www.aclweb.org/anthology/P05-1033.

Chiang, David (2007). "Hierarchical Phrase-Based Translation". In: *Computational Linguistics* 33.2, pp. 201–228. DOI: http://dx.doi.org/10.1162/coli.2007.33.2.201.

Cho, Kyunghyun, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio (2014a). "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)*. Doha, Qatar: Association for Computational Linguistics, pp. 1724–1734. DOI: 10.3115/v1/D14-1179. URL: https://www.aclweb.org/anthology/D14-1179.

Cho, Kyunghyun, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio (2014b). "On the Properties of Neural Machine Translation: Encoder-Decoder Approaches". In: *Proceedings of Eighth Workshop on Syntax, Semantics and*

Structure in Statistical Translation. Doha, Qatar, pp. 103–111. DOI: 10.3115/v1/W14-4012. URL: http://aclweb.org/anthology/W14-4012.

Chomsky, Noam (1957). *Syntactic structure*. Mouton.

Chomsky, Noam (1965). *Aspects of the Theory of Syntax*. Vol. 11. MIT press.

Chomsky, Noam (1981). *Lectures on Government and Binding: The Pisa Lectures*. Holland: Foris Publications. URL: http://www.degruyter.com/view/product/60947.

Chomsky, Noam and Howard Lasnik (1993). "The Theory of Principles and Parameters". In: *Syntax: An International Handbook of Contemporary Research*. Berlin: de Gruyter.

Chu, Yoeng-Jin (1965). "On the shortest arborescence of a directed graph". In: *Science Sinica* 14, pp. 1396–1400.

Clark, Alexander (2001). "Unsupervised language acquisition: Theory and practice". PhD thesis. University of Sussex. URL: https://arxiv.org/pdf/cs/0212024.pdf.

Cohen, Shay and Noah A. Smith (2010). "Viterbi Training for PCFGs: Hardness Results and Competitiveness of Uniform Initialization". In: *Proceedings of ACL*, pp. 1502–1511. URL: http://www.aclweb.org/anthology/P10-1152.

Cohen, Shay B., Kevin Gimpel, and Noah A Smith (2009). "Logistic Normal Priors for Unsupervised Probabilistic Grammar Induction". In: *Advances in Neural Information Processing Systems 21*. Curran Associates, Inc., pp. 321–328.

Cohen, Shay B. and Noah A. Smith (2012). "Empirical Risk Minimization for Probabilistic Grammars: Sample Complexity and Hardness of Learning". In: *Computational Linguistics* 38.3, pp. 479–526. URL: http://www.aclweb.org/anthology/J12-3003.

Collins, Michael (2003). "Head-Driven Statistical Models for Natural Language Parsing". In: *Computational Linguistics* 29.4, pp. 589–637. DOI: 10.1162/089120103322753356. URL: https://www.aclweb.org/anthology/J03-4003.

Collins, Michael and Nigel Duffy (2001). "Convolution Kernels for Natural Language". In: *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*. NIPS'01. Vancouver, British Columbia, Canada: MIT Press, pp. 625–632. URL: http://dl.acm.org/citation.cfm?id=2980539.2980621.

Collins, Michael, Philipp Koehn, and Ivona Kucerova (2005). "Clause Restructuring for Statistical Machine Translation". In: *Proceedings of the 43rd Annual Meeting* *of the Association for Computational Linguistics*. Ann Arbor, Michigan, pp. 531–540. URL: http://aclweb.org/anthology/P05-1066.

Comrie, Bernard (1989). *Language Universals and Linguistic Typology*. Oxford: Basil Blackwell.

Conneau, Alexis, German Kruszewski, Guillaume Lample, Loïc Barrault, and Marco Baroni (2018). "What you can cram into a single $&!#* vector: Probing sentence embeddings for linguistic properties". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, pp. 2126–2136. URL: https://www.aclweb.org/anthology/P18-1198.

Cotterell, Ryan, Nanyun Peng, and Jason Eisner (2015). "Modeling Word Forms Using Latent Underlying Morphs and Phonology". In: *Transactions of the Association for Computational Linguistics* 3, pp. 433–447. URL: http://cs.jhu.edu/~jason/papers/#cotterell-peng-eisner-2015.

Croft, William (2002). *Typology and Universals*. Cambridge University Press. URL: http://www.cambridge.org/us/academic/subjects/languages-linguistics/grammar-and-syntax/typology-and-universals-2nd-edition?format=PB&isbn=9780521004992.

Cross, James and Liang Huang (2016). "Incremental Parsing with Minimal Features Using Bi-Directional LSTM". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Berlin, pp. 32–37. URL: http://anthology.aclweb.org/P16-2006.

Cui, Xiaodong, Vaibhava Goel, and Brian Kingsbury (2015). "Data Augmentation for Deep Neural Network Acoustic Modeling". In: *IEEE/ACM Transactions on Audio, Speech and Language Processing* 23.9, pp. 1469–1477. ISSN: 2329-9290. DOI: 10.1109/TASLP.2015.2438544. URL: http://dx.doi.org/10.1109/TASLP.2015.2438544.

Culotta, Aron and Jeffrey Sorensen (2004). "Dependency Tree Kernels for Relation Extraction". In: *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*. Barcelona, Spain, pp. 423–429. DOI: 10.3115/1218955.1219009. URL: https://www.aclweb.org/anthology/P04-1054.

Daumé III, Hal (2009). "Non-Parametric Bayesian Areal Linguistics". In: *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Boulder, Colorado, pp. 593–601. URL: http://aclweb.org/anthology/N09-1067.

Daumé III, Hal and Lyle Campbell (2007). "A Bayesian Model for Discovering Typological Implications". In: *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*. Prague, Czech Republic, pp. 65–72. URL: http://aclweb.org/anthology/P07-1009.

De Marneffe, Marie-Catherine (2012). "What's that Supposed to Mean? Modeling the Pragmatic Meaning of Utterances". PhD thesis. Stanford University.

Dempster, A. P., N. M. Laird, and D. B. Rubin (1977). "Maximum Likelihood from Incomplete Data via the EM Algorithm". In: *J. Royal Statist. Soc. Ser. B* 39.1, pp. 1–38.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova (2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 4171–4186. URL: https://www.aclweb.org/anthology/N19-1423.

Dorr, Bonnie J. (1993). *Machine Translation: A View from the Lexicon*. Cambridge, MA: MIT Press. URL: https://mitpress.mit.edu/books/machine-translation.

Dorr, Bonnie J., Lisa Pearl, Rebecca Hwa, and Nizar Habash (2002). "DUSTer: A Method for Unraveling Cross-Language Divergences for Statistical Word-Level Alignment". In: *Proceedings of the 5th Conference of the Association for Machine Translation in the Americas on Machine Translation: From Research to Real Users*. AMTA '02. London, UK, UK, pp. 31–43. URL: http://dl.acm.org/citation.cfm?id=648181.749385.

Dozat, Timothy and Christopher D. Manning (2017). "Deep Biaffine Attention for Neural Dependency Parsing". In: *Proceedings of the International Conference on Learning Representations*. URL: https://web.stanford.edu/~tdozat/files/TDozat-ICLR2017-Paper.pdf.

Drucker, Harris, Christopher JC Burges, Linda Kaufman, Alex J Smola, and Vladimir Vapnik (1997). "Support vector regression machines". In: *Advances in neural information processing systems*, pp. 155–161.

Dryer, Matthew S. and Martin Haspelmath, eds. (2013). *The World Atlas of Language Structures Online*. Leipzig: Max Planck Institute for Evolutionary Anthropology. URL: http://wals.info/.

Duong, Long, Trevor Cohn, Steven Bird, and Paul Cook (2015a). “Cross-lingual Transfer for Unsupervised Dependency Parsing Without Parallel Data”. In: *Proceedings of the Nineteenth Conference on Computational Natural Language Learning*. Beijing, China: Association for Computational Linguistics, pp. 113–122. DOI: 10.18653/v1/K15-1012. URL: https://www.aclweb.org/anthology/K15-1012.

Duong, Long, Trevor Cohn, Steven Bird, and Paul Cook (2015b). “Cross-lingual Transfer for Unsupervised Dependency Parsing Without Parallel Data”. In: *Proceedings of the 19th Conference on Computational Natural Language Learning*. Beijing, China, pp. 113–122. URL: http://aclweb.org/anthology/K15-1012.

Dyer, Chris, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith (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)*. Beijing, China: Association for Computational Linguistics, pp. 334–343. DOI: 10.3115/v1/P15-1033. URL: https://www.aclweb.org/anthology/P15-1033.

Edmonds, Jack (1967). “Optimum branchings”. In: *Journal of Research of the National Bureau of Standards B* 71.4, pp. 233–240.

Eisner, Jason (1996). “Three New Probabilistic Models for Dependency Parsing: An Exploration”. In: *Proceedings of the 16th International Conference on Computational Linguistics*, pp. 340–345. URL: http://cs.jhu.edu/~jason/papers/#eisner-1996-coling.

Eisner, Jason (2000). “Bilexical Grammars and Their Cubic-Time Parsing Algorithms”. In: *Advances in Probabilistic and Other Parsing Technologies*. Ed. by Harry Bunt and Anton Nijholt. Kluwer Academic Publishers, pp. 29–62. URL: http://cs.jhu.edu/~jason/papers/#eisner-2000-iwptbook.

Eisner, Jason (2003). “Learning Non-Isomorphic Tree Mappings for Machine Translation”. In: *The Companion Volume to the Proceedings of 41st Annual Meeting of the Association for Computational Linguistics*. Sapporo, Japan: Association for Computational Linguistics, pp. 205–208. DOI: 10.3115/1075178.1075217. URL: https://www.aclweb.org/anthology/P03-2041.

Eisner, Jason (2016). “Inside-Outside and Forward-Backward Algorithms are Just Backprop”. In: *Proceedings of the EMNLP Workshop on Structured Prediction for NLP*. Austin, TX, pp. 1–17. URL: http://cs.jhu.edu/~jason/papers/#eisner-2016.

Eisner, Jason and Damianos Karakos (2005). “Bootstrapping Without the Boot”. In: *Proceedings of Human Language Technology Conference and Conference on*

Empirical Methods in Natural Language Processing, pp. 395–402. URL: http://www.aclweb.org/anthology/H/H05/H05-1050.

Eisner, Jason and Giorgio Satta (1999). "Efficient Parsing for Bilexical Context-Free Grammars and Head Automaton Grammars". In: *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*. College Park, Maryland, USA: Association for Computational Linguistics, pp. 457–464. DOI: 10.3115/1034678.1034748. URL: https://www.aclweb.org/anthology/P99-1059.

Eisner, Jason and Noah A. Smith (2010). "Favor Short Dependencies: Parsing with Soft and Hard Constraints on Dependency Length". In: *Trends in Parsing Technology: Dependency Parsing, Domain Adaptation, and Deep Parsing*. Ed. by Harry Bunt, Paola Merlo, and Joakim Nivre. Chap. 8, pp. 121–150. URL: http://cs.jhu.edu/~jason/papers/#eisner-smith-2010-iwptbook.

Eisner, Jason and Roy W. Tromble (2006). "Local Search with Very Large-Scale Neighborhoods for Optimal Permutations in Machine Translation". In: *Proceedings of the HLT-NAACL Workshop on Computationally Hard Problems and Joint Inference in Speech and Language Processing*, pp. 57–75. URL: http://cs.jhu.edu/~jason/papers/#eisner-tromble-2006.

Fernández-González, Daniel and Carlos Gómez-Rodríguez (2019). "Left-to-Right Dependency Parsing with Pointer Networks". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 710–716. DOI: 10.18653/v1/N19-1076. URL: https://www.aclweb.org/anthology/N19-1076.

Filippova, Katja and Michael Strube (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*, pp. 225–228. URL: http://www.aclweb.org/anthology/N/N09/N09-2057.

Fisch, Adam, Jiang Guo, and Regina Barzilay (2019). "Working Hard or Hardly Working: Challenges of Integrating Typology into Neural Dependency Parsers". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Frank, Robert and Shyam Kapur (1996). "On the Use of Triggers in Parameter Setting". In: *Linguistic Inquiry* 27, pp. 623–660. URL: https://www.jstor.org/stable/4178955.

Fu, King-Sun and Taylor L Booth (1975). “Grammatical inference: Introduction and survey-Part I”. In: *IEEE Transactions on Systems, Man, and Cybernetics* 1, pp. 95–111.

Futrell, Richard and Edward Gibson (2015). “Experiments with Generative Models for Dependency Tree Linearization”. In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 1978–1983. URL: http://aclweb.org/anthology/D15-1231.

Futrell, Richard, Kyle Mahowald, and Edward Gibson (2015). “Quantifying Word Order Freedom in Dependency Corpora”. In: *Proceedings of the Third International Conference on Dependency Linguistics*, pp. 91–100. URL: http://www.aclweb.org/anthology/W15-2112.

Galley, Michel, Mark Hopkins, Kevin Knight, and Daniel Marcu (2004). “What’s in a translation rule?” In: *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT-NAACL 2004*. Boston, Massachusetts, USA: Association for Computational Linguistics, pp. 273–280. URL: https://www.aclweb.org/anthology/N04-1035.

Ganchev, Kuzman, Jennifer Gillenwater, and Ben Taskar (2009). “Dependency Grammar Induction via Bitext Projection Constraints”. 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*. Suntec, Singapore, pp. 369–377. URL: http://www.aclweb.org/anthology/P09-1042.

Ganchev, Kuzman, Joao Graça, Jennifer Gillenwater, and Ben Taskar (2010). “Posterior regularization for structured latent variable models”. In: *Journal of Machine Learning Research* 11, pp. 2001–2049. URL: http://www.jmlr.org/papers/v11/ganchev10a.html.

Georgi, Ryan, Fei Xia, and William Lewis (2010). “Comparing Language Similarity across Genetic and Typologically-Based Groupings”. In: *Proceedings of the 23rd International Conference on Computational Linguistics*. Beijing, China, pp. 385–393. URL: http://aclweb.org/anthology/C10-1044.

Gershman, Samuel and Noah Goodman (2014). “Amortized inference in probabilistic reasoning”. In: *Proceedings of the annual meeting of the cognitive science society*. Vol. 36. 36.

Gibson, Edward and Kenneth Wexler (1994). “Triggers”. In: *Linguistic Inquiry* 25.3, pp. 407–454. URL: http://www.jstor.org/stable/4178869.

Gildea, Daniel and Martha Palmer (2002). “The Necessity of Parsing for Predicate Argument Recognition”. In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, pp. 239–246. DOI: 10.3115/1073083.1073124. URL: https://www.aclweb.org/anthology/P02-1031.

Gillenwater, Jennifer, Kuzman Ganchev, João Graça, Fernando Pereira, and Ben Taskar (2010). “Sparsity in Dependency Grammar Induction”. In: *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL 2010) Short Papers*. Uppsala, Sweden: Association for Computational Linguistics, pp. 194–199. URL: https://www.aclweb.org/anthology/P10-2036.

Gimpel, Kevin and Noah A. Smith (2012). “Concavity and Initialization for Unsupervised Dependency Parsing”. In: *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Montréal, Canada: Association for Computational Linguistics, pp. 577–581. URL: https://www.aclweb.org/anthology/N12-1069.

Glorot, Xavier and Yoshua Bengio (2010). “Understanding the difficulty of training deep feedforward neural networks”. In: *Proceedings of the International Conference on Artificial Intelligence and Statistics*. URL: http://jmlr.org/proceedings/papers/v9/glorot10a/glorot10a.pdf.

Goller, Christoph and Alexander Kuchler (1996). “Learning task-dependent distributed representations by backpropagation through structure”. In:

Gormley, Matthew R. and Jason Eisner (2013). “Nonconvex Global Optimization for Latent-Variable Models”. In: *ACL*. Sofia, Bulgaria. URL: http://cs.jhu.edu/~jason/papers/#gormley-eisner-2013.

Grandvalet, Yves and Yoshua Bengio (2005). “Semi-supervised Learning by Entropy Minimization”. In: *Advances in Neural Information Processing Systems 17*. Ed. by L. K. Saul, Y. Weiss, and L. Bottou. MIT Press, pp. 529–536. URL: http://papers.nips.cc/paper/2740-semi-supervised-learning-by-entropy-minimization.pdf.

Grave, Edouard and Noémie Elhadad (2015). “A convex and feature-rich discriminative approach to dependency grammar induction”. In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics, ACL 2015, July 26-31, 2015, Beijing, China*, pp. 1375–1384.

Graves, Alex (2012). *Supervised Sequence Labelling with Recurrent Neural Networks*. Springer. URL: http://www.cs.toronto.edu/~graves/preprint.pdf.

Greenberg, Joseph H. (1963). “Some Universals of Grammar with Particular Reference to the Order of Meaningful Elements”. In: *Universals of Language*. Ed. by Joseph H. Greenberg. MIT Press, pp. 73–113.

Gulordava, Kristina and Paola Merlo (2016). “Multi-lingual Dependency Parsing Evaluation: A Large-scale Analysis of Word Order Properties using Artificial Data”. In: *Transactions of the Association for Computational Linguistics* 4, pp. 343–356. URL: http://aclweb.org/anthology/Q16-1025.

Guo, Jiang, Wanxiang Che, David Yarowsky, Haifeng Wang, and Ting Liu (2015). “Cross-lingual Dependency Parsing Based on Distributed Representations”. In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*. Beijing, China, pp. 1234–1244. URL: http://aclweb.org/anthology/P15-1119.

Guo, Jiang, Wanxiang Che, David Yarowsky, Haifeng Wang, and Ting Liu (2016). “A Representation Learning Framework for Multi-source Transfer Parsing”. In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*. AAAI’16. Phoenix, Arizona: AAAI Press, pp. 2734–2740. URL: http://dl.acm.org/citation.cfm?id=3016100.3016284.

Hamilton, Will, Payal Bajaj, Marinka Zitnik, Dan Jurafsky, and Jure Leskovec (2018). “Embedding Logical Queries on Knowledge Graphs”. In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Curran Associates, Inc., pp. 2026–2037. URL: http://papers.nips.cc/paper/7473-embedding-logical-queries-on-knowledge-graphs.pdf.

Hawkins, John (1994). *A Performance Theory of Order and Constituency*. Cambridge University Press. URL: http://www.cambridge.org/us/academic/subjects/languages-linguistics/grammar-and-syntax/performance-theory-order-and-constituency.

Hawkins, John A. (2014). *Word Order Universals*. Elsevier. URL: https://www.elsevier.com/books/word-order-universals/hawkins/978-0-12-333370-4.

Headden III, William P., Mark Johnson, and David McClosky (2009). “Improving Unsupervised Dependency Parsing with Richer Contexts and Smoothing”. In: *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, pp. 101–109. URL: http://www.aclweb.org/anthology/N/N09/N09-1012.

Hellwig, Peter (1986). “Dependency unification grammar”. In: *Proceedings of the 11th International Conference on Computational Linguistics*. Vol. 1.

Hermann, Karl Moritz, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom (2015). “Teaching Machines to Read and Comprehend”. In: *Advances in Neural Information Processing Systems*, pp. 1684–1692. URL: http://arxiv.org/abs/1506.03340.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). “Long Short-Term Memory”. In: *Neural Computation* 9.8, pp. 1735–1780. URL: http://www.mitpressjournals.org/doi/abs/10.1162/neco.1997.9.8.1735#.WKp_mxLyso9.

Howlett, Susan and Mark Dras (2011). “Clause Restructuring For SMT Not Absolutely Helpful”. In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA, pp. 384–388. URL: http://aclweb.org/anthology/P11-2067.

Huang, Zhongqiang, Martin Čmejrek, and Bowen Zhou (2010). “Soft Syntactic Constraints for Hierarchical Phrase-Based Translation Using Latent Syntactic Distributions”. In: *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. Cambridge, MA: Association for Computational Linguistics, pp. 138–147. URL: https://www.aclweb.org/anthology/D10-1014.

Hudson, Richard A (1984). *Word grammar*. Blackwell Oxford.

Hwa, Rebecca, Philip Resnik, Amy Weinberg, Clara Cabezas, and Okan Kolak (2005). “Bootstrapping Parsers via Syntactic Projection Across Parallel Texts”. In: *Natural Language Engineering* 11.3, pp. 311–325. ISSN: 1351-3249. DOI: 10.1017/S1351324905003840. URL: http://dx.doi.org/10.1017/S1351324905003840.

Jaitly, Navdeep and Geoffrey E. Hinton (2013). “Vocal Tract Length Perturbation (VTLP) Improves Speech Recognition”. In: *Proceedings of the 30th International Conference on Machine Learning Workshop on Deep Learning for Audio, Speech and Language*. URL: https://www.cs.toronto.edu/~hinton/absps/perturb.pdf.

Jiang, Yong, Wenjuan Han, and Kewei Tu (2017). “Combining Generative and Discriminative Approaches to Unsupervised Dependency Parsing via Dual Decomposition”. In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 1689–1694. URL: http://aclweb.org/anthology/D17-1177.

Johnson, Mark, Thomas Griffiths, and Sharon Goldwater (2007). “Bayesian Inference for PCFGs via Markov Chain Monte Carlo”. In: *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for*

Computational Linguistics; Proceedings of the Main Conference. Rochester, New York: Association for Computational Linguistics, pp. 139–146. URL: https://www.aclweb.org/anthology/N07-1018.

Kahane, Sylvain (2012). "Why to choose dependency rather than constituency for syntax: a formal point of view". In: *J. Apresjan, M.-C. L'Homme, M.-C. Iomdin, J. Milicevic, A. Polguère, and L. Wanner, editors, Meanings, Texts, and other exciting things: A Festschrift to Commemorate the 80th Anniversary of Professor Igor A. MelâĂŹcuk*, pp. 257–272.

Karakos, Damianos, Jason Eisner, Sanjeev Khudanpur, and Carey E. Priebe (2007). "Cross-Instance Tuning of Unsupervised Document Clustering Algorithms". In: *Human Language Technologies: Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 252–259. URL: http://www.aclweb.org/anthology/N/N07/N07-1032.

Kim, Joo-Kyung, Young-Bum Kim, Ruhi Sarikaya, and Eric Fosler-Lussier (2017). "Cross-Lingual Transfer Learning for POS Tagging without Cross-Lingual Resources". In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark, pp. 2832–2838. URL: http://aclweb.org/anthology/D17-1302.

Kingma, Diederik and Jimmy Ba (2015). "Adam: A Method for Stochastic Optimization". In: *Proceedings of the International Conference on Learning Representations*. URL: https://arxiv.org/pdf/1412.6980.pdf.

Kiperwasser, Eliyahu and Yoav Goldberg (2016). "Simple and Accurate Dependency Parsing Using Bidirectional LSTM Feature Representations". In: *Transactions of the Association for Computational Linguistics* 4, pp. 313–327. DOI: 10.1162/tacl_a_00101. URL: https://www.aclweb.org/anthology/Q16-1023.

Klein, Dan and Christopher Manning (2004). "Corpus-Based Induction of Syntactic Structure: Models of Dependency and Constituency". In: *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics*, pp. 478–485. DOI: 10.3115/1218955.1219016. URL: http://www.aclweb.org/anthology/P04-1061.

Klein, Dan and Christopher D Manning (2005). "The unsupervised learning of natural language structure". PhD thesis.

Kong, Lingpeng, Alexander M. Rush, and Noah A. Smith (2015). "Transforming Dependencies into Phrase Structures". In: *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Denver, Colorado: Association for Computational

Linguistics, pp. 788–798. DOI: 10.3115/v1/N15-1080. URL: https://www.aclweb.org/anthology/N15-1080.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton (2012). “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, pp. 1097–1105.

Kruijff, Geert-Jan M (2002). “Formal and computational aspects of dependency grammar: History and development of DG”. In: *Tech. Report, ESSLLI*.

Kurihara, Kenichi and Taisuke Sato (2004). “An application of the variational Bayesian approach to probabilistic context-free grammars”. In: *IJCNLP-04 Workshop beyond shallow analyses*.

Lacroix, Ophélie, Lauriane Aufrant, Guillaume Wisniewski, and François Yvon (2016). “Frustratingly Easy Cross-Lingual Transfer for Transition-Based Dependency Parsing”. In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. San Diego, California, pp. 1058–1063. DOI: 10.18653/v1/N16-1121. URL: http://www.aclweb.org/anthology/N16-1121.

Lari, Karim and Steve J. Young (1990). “The estimation of stochastic context-free grammars using the Inside-Outside algorithm”. In: *Computer Speech and Language* 4.1, pp. 35–56. URL: http://www.sciencedirect.com/science/article/pii/088523089090022X.

Le, Phong and Willem Zuidema (2015). “Unsupervised Dependency Parsing: Let’s Use Supervised Parsers”. In: *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 651–661. URL: http://www.aclweb.org/anthology/N15-1067.

LeCun, Yann, Sumit Chopra, Raia Hadsell, and Fu Jie Huang (2007). “A Tutorial on Energy-Based Learning”. In: *Predicting Structured Data*. Ed. by Gökhan Bakır, Thomas Hofmann, Bernhard Schölkopf, Alexander J. Smola, Ben Taskar, and S. V. N. Vishwanathan. MIT Press. URL: http://yann.lecun.com/exdb/publis/orig/lecun-06.pdf.

Lee, Juho, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh (2019). “Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks”. In: *International Conference on Machine Learning*, pp. 3744–3753.

Lee, Lillian (1999). “Measures of Distributional Similarity”. In: *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pp. 25–32. URL: http://www.aclweb.org/anthology/P99-1004.

Lee, Lillian (2001). “On the effectiveness of the skew divergence for statistical language analysis”. In: *Proceedings of AISTATS*.

Lehmann, Erich L and George Casella (2006). *Theory of point estimation*. Springer Science & Business Media.

Levy, Omer and Yoav Goldberg (2014). “Dependency-Based Word Embeddings”. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Baltimore, Maryland: Association for Computational Linguistics, pp. 302–308. DOI: 10.3115/v1/P14-2050. URL: https://www.aclweb.org/anthology/P14-2050.

Lewis, William D. and Fei Xia (2008). “Automatically Identifying Computationally Relevant Typological Features”. In: *Proceedings of the Third International Joint Conference on Natural Language Processing: Volume-II*. URL: http://aclweb.org/anthology/I08-2093.

Li, Xiang Lisa, Dingquan Wang, and Jason Eisner (2019). “A Generative Model for Punctuation in Dependency Trees”. In: *Transactions of the Association for Computational Linguistics (TACL)*, pp. 357–373. ISSN: 2307-387X. URL: http://cs.jhu.edu/~jason/papers/#li-et-al-2019-tacl.

Lin, Yongjie, Yi Chern Tan, and Robert Frank (2019). “Open Sesame: Getting Inside BERT’s Linguistic Knowledge”. In: *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*.

Linzen, Tal, Emmanuel Dupoux, and Yoav Goldberg (2016). “Assessing the Ability of LSTMs to Learn Syntax-Sensitive Dependencies”. In: *Transactions of the Association for Computational Linguistics* 4, pp. 521–535. DOI: 10.1162/tacl_a_00115. URL: https://www.aclweb.org/anthology/Q16-1037.

Liu, Haitao (2010). “Dependency Direction as a Means of Word-Order Typology: A Method Based on Dependency Treebanks”. In: *Lingua* 120.6, pp. 1567–1578.

Ma, Xuezhe and Fei Xia (2014). “Unsupervised Dependency Parsing with Transferring Distribution via Parallel Guidance and Entropy Regularization”. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*. Vol. 1. Baltimore, Maryland, pp. 1337–1348. URL: http://aclweb.org/anthology/P14-1126.

Ma, Xuezhe, Zecong Hu, Jingzhou Liu, Nanyun Peng, Graham Neubig, and Eduard Hovy (2018). “Stack-Pointer Networks for Dependency Parsing”. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, pp. 1403–1414. DOI: 10.18653/v1/P18-1130. URL: https://www.aclweb.org/anthology/P18-1130.

Majliš, Martin (2011). *W2C—Web to Corpus—Corpora*. URL: http://hdl.handle.net/11858/00-097C-0000-0022-6133-9.

Malaviya, Chaitanya, Graham Neubig, and Patrick Littell (2017). "Learning Language Representations for Typology Prediction". In: *Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Copenhagen, Denmark. URL: https://www.phontron.com/paper/malaviya17emnlp.pdf.

Marcheggiani, Diego and Ivan Titov (2017). "Encoding Sentences with Graph Convolutional Networks for Semantic Role Labeling". In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, pp. 1506–1515. DOI: 10.18653/v1/D17-1159. URL: https://www.aclweb.org/anthology/D17-1159.

Marcus, Mitchell P., Mary Ann Marcinkiewicz, and Beatrice Santorini (1993). "Building a large annotated corpus of English: The Penn Treebank". In: *Computational linguistics* 19.2, pp. 313–330. URL: http://dl.acm.org/citation.cfm?id=972475 (visited on 11/09/2013).

Mareček, David and Milan Straka (2013). "Stop-probability estimates computed on a large corpus improve Unsupervised Dependency Parsing". In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Sofia, Bulgaria, pp. 281–290. URL: http://aclweb.org/anthology/P13-1028.

Mareček, David (2016). "Twelve Years of Unsupervised Dependency Parsing". In: *Proceedings of the 16th ITAT Conference Information Technologies—Applications and Theory*, pp. 56–62. URL: http://ceur-ws.org/Vol-1649/56.pdf.

Marneffe, Marie-Catherine de, Timothy Dozat, Natalia Silveira, Katri Haverinen, Filip Ginter, Joakim Nivre, and Christopher D. Manning (2014). "Universal Stanford dependencies: A cross-linguistic typology". In: *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*. Reykjavik, Iceland: European Language Resources Association (ELRA), pp. 4585–4592. URL: http://www.lrec-conf.org/proceedings/lrec2014/pdf/1062_Paper.pdf.

Martínez Alonso, Héctor, Željko Agić, Barbara Plank, and Anders Søgaard (2017). "Parsing Universal Dependencies without training". In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pp. 230–240. URL: http://aclanthology.coli.uni-saarland.de/pdf/E/E17/E17-1022.pdf.

Maxwell, Dan (2013). "Why So Many Nodes?" In: *Proceedings of the Second International Conference on Dependency Linguistics (DepLing 2013)*. Prague, Czech

Republic: Charles University in Prague, Matfyzpress, Prague, Czech Republic, pp. 197–206. URL: https://www.aclweb.org/anthology/W13-3722.

McCann, Bryan, James Bradbury, Caiming Xiong, and Richard Socher (2017). "Learned in Translation: Contextualized Word Vectors". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., pp. 6294–6305. URL: http://papers.nips.cc/paper/7209-learned-in-translation-contextualized-word-vectors.pdf.

McDonald, Ryan (2006). "Discriminative Learning and Spanning Tree Algorithms for Dependency Parsing". PhD thesis. University of Pennsylvania. URL: https://repository.upenn.edu/dissertations/AAI3225503/.

McDonald, Ryan, Koby Crammer, and Fernando Pereira (2005). "Online Large-Margin Training of Dependency Parsers". In: *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pp. 91–98. URL: http://aclanthology.coli.uni-saarland.de/pdf/P/P05/P05-1012.pdf.

McDonald, Ryan, Slav Petrov, and Keith Hall (2011). "Multi-Source Transfer of Delexicalized Dependency Parsers". In: *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*. Edinburgh, Scotland, UK., pp. 62–72. URL: http://aclweb.org/anthology/D11-1006.

McDonald, Ryan, Fernando Pereira, Kiril Ribarov, and Jan Hajič (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*. Vancouver, British Columbia, Canada: Association for Computational Linguistics, pp. 523–530. URL: https://www.aclweb.org/anthology/H05-1066.

McDonald, Ryan, Joakim Nivre, Yvonne Quirmbach-Brundage, Yoav Goldberg, Dipanjan Das, Kuzman Ganchev, Keith Hall, Slav Petrov, Hao Zhang, Oscar Täckström, Claudia Bedini, Núria Bertomeu Castelló, and Jungmee Lee (2013). "Universal Dependency Annotation for Multilingual Parsing". In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Sofia, Bulgaria, pp. 92–97. URL: http://www.aclweb.org/anthology/P13-2017.

Mel'cuk, Igor Aleksandrovic et al. (1988). *Dependency syntax: theory and practice*. SUNY press.

Mihalcea, Rada and Paul Tarau (2004). "TextRank: Bringing Order into Text". In: *Proceedings of the 2004 Conference on Empirical Methods in Natural Language*

Processing. Barcelona, Spain: Association for Computational Linguistics, pp. 404–411. URL: https://www.aclweb.org/anthology/W04-3252.

Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean (2013). "Distributed representations of words and phrases and their compositionality". In: *Advances in Neural Information Processing Systems*, pp. 3111–3119.

Mille, Simon, Anja Belz, Bernd Bohnet, Yvette Graham, Emily Pitler, and Leo Wanner (2018). "The First Multilingual Surface Realisation Shared Task (SR'18): Overview and Evaluation Results". In: *Proceedings of the 1st Workshop on Multilingual Surface Realization (MSR), 56th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 1–12. URL: http://www.aclweb.org/anthology/W18-3601.

Murawaki, Yugo (2015). "Continuous Space Representations of Linguistic Typology and their Application to Phylogenetic Inference". In: *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Denver, Colorado, pp. 324–334. DOI: 10.3115/v1/N15-1036. URL: http://aclweb.org/anthology/N15-1036.

Naseem, Tahira, Regina Barzilay, and Amir Globerson (2012). "Selective Sharing for Multilingual Dependency Parsing". In: *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Jeju Island, Korea, pp. 629–637. URL: http://aclweb.org/anthology/P12-1066.

Naseem, Tahira, Harr Chen, Regina Barzilay, and Mark Johnson (2010). "Using Universal Linguistic Knowledge to Guide Grammar Induction". In: *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. Cambridge, MA, pp. 1234–1244. URL: http://aclweb.org/anthology/D10-1120.

Nguyen, Dat Quoc, Dai Quoc Nguyen, Dang Duc Pham, and Son Bao Pham (2014). "RDRPOSTagger: A Ripple Down Rules-based Part-Of-Speech Tagger". In: *Proceedings of the Demonstrations at the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 17–20. URL: http://www.aclweb.org/anthology/E14-2005.

Nivre, Joakim (2003). "An Efficient Algorithm for Projective Dependency Parsing". In: *Proceedings of the Eighth International Workshop on Parsing Technologies (IWPT*. Nancy, France, pp. 149–160. URL: https://www.aclweb.org/anthology/W03-3017.

Nivre, Joakim (2005). "Dependency grammar and dependency parsing". In: *MSI report* 5133.1959, pp. 1–32.

Nivre, Joakim (2008). “Algorithms for Deterministic Incremental Dependency Parsing”. In: *Computational Linguistics* 34.4, pp. 513–553. URL: https://www.mitpressjournals.org/doi/pdf/10.1162/coli.07-056-R1-07-027.

Nivre, Joakim and Jens Nilsson (2005). “Pseudo-Projective Dependency Parsing”. In: *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*. Ann Arbor, Michigan, pp. 99–106. URL: http://aclweb.org/anthology/P05-1013.

Nivre, Joakim, Željko Agić, Maria Jesus Aranzabe, Masayuki Asahara, Aitziber Atutxa, Miguel Ballesteros, John Bauer, Kepa Bengoetxea, Riyaz Ahmad Bhat, Cristina Bosco, Sam Bowman, Giuseppe G. A. Celano, Miriam Connor, Marie-Catherine de Marneffe, Arantza Diaz de Ilarraza, Kaja Dobrovoljc, Timothy Dozat, Tomaž Erjavec, Richárd Farkas, Jennifer Foster, Daniel Galbraith, Filip Ginter, Iakes Goenaga, Koldo Gojenola, Yoav Goldberg, Berta Gonzales, Bruno Guillaume, Jan Hajič, Dag Haug, Radu Ion, Elena Irimia, Anders Johannsen, Hiroshi Kanayama, Jenna Kanerva, Simon Krek, Veronika Laippala, Alessandro Lenci, Nikola Ljubešić, Teresa Lynn, Christopher Manning, CĂČtĂČlina MĂČrĂČnduc, David Mareček, Héctor Martínez Alonso, Jan Mašek, Yuji Matsumoto, Ryan McDonald, Anna Missilä, Verginica Mititelu, Yusuke Miyao, Simonetta Montemagni, Shunsuke Mori, Hanna Nurmi, Petya Osenova, Lilja Øvrelid, Elena Pascual, Marco Passarotti, Cenel-Augusto Perez, Slav Petrov, Jussi Piitulainen, Barbara Plank, Martin Popel, Prokopis Prokopidis, Sampo Pyysalo, Loganathan Ramasamy, Rudolf Rosa, Shadi Saleh, Sebastian Schuster, Wolfgang Seeker, Mojgan Seraji, Natalia Silveira, Maria Simi, Radu Simionescu, Katalin Simkó, Kiril Simov, Aaron Smith, Jan Štěpánek, Alane Suhr, Zsolt Szántó, Takaaki Tanaka, Reut Tsarfaty, Sumire Uematsu, Larraitz Uria, Viktor Varga, Veronika Vincze, Zdeněk Žabokrtský, Daniel Zeman, and Hanzhi Zhu (2015). *Universal Dependencies 1.2*. URL: http://hdl.handle.net/11234/1-1548.

Nivre, Joakim, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajič, Christopher Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Reut Tsarfaty, and Daniel Zeman (2016). “Universal Dependencies v1: A Multilingual Treebank Collection”. In: *Proceedings of the 10th International Conference on Language Resources and Evaluation*, pp. 1659–1666. URL: http://www.lrec-conf.org/proceedings/lrec2016/pdf/348_Paper.pdf.

Nivre, Joakim et al. (2019). *Universal Dependencies 2.4*. URL: http://hdl.handle.net/11234/1-2988.

Noji, Hiroshi, Yusuke Miyao, and Mark Johnson (2016). “Using Left-corner Parsing to Encode Universal Structural Constraints in Grammar Induction”. In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, Texas, pp. 33–43. URL: http://aclweb.org/anthology/D16-1004.

Nunberg, Geoffrey (1990). *The Linguistics of Punctuation*. CSLI Lecture Notes 18. Center for the Study of Language and Information. URL: https://web.stanford.edu/group/cslipublications/cslipublications/site/0937073466.shtml.

O’Horan, Helen, Yevgeni Berzak, Ivan Vulic, Roi Reichart, and Anna Korhonen (2016). “Survey on the Use of Typological Information in Natural Language Processing”. In: *Proceedings of the 26th International Conference on Computational Linguistics: Technical Papers*. Osaka, Japan, pp. 1297–1308. URL: http://aclweb.org/anthology/C16-1123.

Östling, Robert (2015). “Word Order Typology through Multilingual Word Alignment”. 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)*. Beijing, China: Association for Computational Linguistics, pp. 205–211. DOI: 10.3115/v1/P15-2034.

Page, Lawrence, Sergey Brin, Rajeev Motwani, and Terry Winograd (1999). *The PageRank Citation Ranking: Bringing Order to the Web*. Technical Report 1999-66. Stanford InfoLab. URL: http://ilpubs.stanford.edu:8090/422/.

Pakman, Ari and Liam Paninski (2018). “Amortized Bayesian inference for clustering models”. In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc.

Paskin, Mark A (2001). *Cubic-time parsing and learning algorithms for grammatical bigram models*. Citeseer.

Paskin, Mark A (2002). “Grammatical digrams”. In: *Advances in Neural Information Processing Systems* 14.1, pp. 91–97.

Paszke, Adam, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer (2017). “Automatic differentiation in PyTorch”. In:

Pate, John K and Mark Johnson (2016). “Grammar induction from (lots of) words alone”. In: *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*. Osaka, Japan: The COLING 2016 Organizing Committee, pp. 23–32. URL: https://www.aclweb.org/anthology/C16-1003.

Peters, Matthew, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer (2018). "Deep Contextualized Word Representations". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, pp. 2227–2237. DOI: 10.18653/v1/N18-1202. URL: https://www.aclweb.org/anthology/N18-1202.

Platanios, Emmanouil Antonios, Mrinmaya Sachan, Graham Neubig, and Tom Mitchell (2018). "Contextual Parameter Generation for Universal Neural Machine Translation". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, pp. 425–435. DOI: 10.18653/v1/D18-1039.

Puzikov, Yevgeniy and Iryna Gurevych (2018). "BinLin: A Simple Method of Dependency Tree Linearization". In: *Proceedings of the First Workshop on Multilingual Surface Realisation*, pp. 13–28. URL: http://www.aclweb.org/anthology/W18-3602.

Qian, Peng, Xipeng Qiu, and Xuanjing Huang (2016). "Analyzing Linguistic Knowledge in Sequential Model of Sentence". In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, Texas: Association for Computational Linguistics, pp. 826–835. DOI: 10.18653/v1/D16-1079. URL: https://www.aclweb.org/anthology/D16-1079.

Radford, Alec, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever (2019). "Language models are unsupervised multitask learners". In: *OpenAI Blog* 1.8.

Rajpurkar, Pranav, Jian Zhang, Konstantin Lopyrev, and Percy Liang (2016). "SQuAD: 100,000+ Questions for Machine Comprehension of Text". In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, Texas, pp. 2383–2392. DOI: 10.18653/v1/D16-1264. URL: http://www.aclweb.org/anthology/D16-1264.

Rasooli, Mohammad Sadegh and Michael Collins (2015). "Density-Driven Cross-Lingual Transfer of Dependency Parsers". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 328–338. URL: http://aclweb.org/anthology/D15-1039.

Rasooli, Mohammad Sadegh and Michael Collins (2017). "Cross-Lingual Syntactic Transfer with Limited Resources". In: *Transactions of the Association for Computational Linguistics* 5, pp. 279–293. ISSN: 2307-387X. URL: https://transacl.org/ojs/index.php/tacl/article/view/922.

Rasooli, Mohammad Sadegh and Joel R. Tetreault (2015). "Yara Parser: A Fast and Accurate Dependency Parser". In: *Computing Research Repository* arXiv:1503.06733. URL: http://arxiv.org/abs/1503.06733.

Rasooli, Mohammad Sadegh, Thomas Lippincott, Nizar Habash, and Owen Rambow (2014). "Unsupervised Morphology-Based Vocabulary Expansion". In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1349–1359. URL: http://aclweb.org/anthology/P14-1127.

Ravfogel, Shauli, Yoav Goldberg, and Tal Linzen (2019). "Studying the Inductive Biases of RNNs with Synthetic Variations of Natural Languages". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 3532–3542. URL: https://www.aclweb.org/anthology/N19-1356.

Rosa, Rudolf and David Mareček (2018). "CUNI x-ling: Parsing Under-Resourced Languages in CoNLL 2018 UD Shared Task". In: *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*. Brussels, Belgium: Association for Computational Linguistics, pp. 187–196. DOI: 10.18653/v1/K18-2019. URL: https://www.aclweb.org/anthology/K18-2019.

Rosa, Rudolf and Zdeněk Žabokrtský (2015a). "KLcpos3 — A Language Similarity Measure for Delexicalized Parser Transfer". 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)*. Beijing, China, pp. 243–249. DOI: 10.3115/v1/P15-2040. URL: http://www.aclweb.org/anthology/P15-2040.

Rosa, Rudolf and Zdeněk Žabokrtský (2015b). "MSTParser Model Interpolation for Multi-Source Delexicalized Transfer". In: *Proceedings of the 14th International Conference on Parsing Technologies*, pp. 71–75. DOI: 10.18653/v1/W15-2209. URL: http://www.aclweb.org/anthology/W15-2209.

Roth, Michael and Mirella Lapata (2016). "Neural Semantic Role Labeling with Dependency Path Embeddings". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, pp. 1192–1202. DOI: 10.18653/v1/P16-1113. URL: https://www.aclweb.org/anthology/P16-1113.

Rozovskaya, Alla and Dan Roth (2010). “Training Paradigms for Correcting Errors in Grammar and Usage”. In: *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Los Angeles, California, pp. 154–162. URL: http://aclweb.org/anthology/N10-1018.

Ruder, Sebastian, Ivan Vulić, and Anders Søgaard (2017). “A Survey Of Cross-Lingual Word Embedding Models”. In: *Computing Research Repository* arXiv:1706.04902. URL: https://arxiv.org/abs/1706.04902.

Saha Roy, Rishiraj, Rahul Katare, Niloy Ganguly, and Monojit Choudhury (2014). “Automatic Discovery of Adposition Typology”. In: *Proceedings of the 25th International Conference on Computational Linguistics: Technical Papers*. Dublin, Ireland, pp. 1037–1046. URL: http://aclweb.org/anthology/C14-1098.

Sakakibara, Yasubumi, Michael Brown, Richard Hughey, I Saira Mian, Kimmen Sjölander, Rebecca C Underwood, and David Haussler (1994). “Stochastic context-free grammers for tRNA modeling”. In: *Nucleic acids research* 22.23, pp. 5112–5120.

Sedgewick, Robert (1977). “Permutation Generation Methods”. In: *ACM Computing Surveys* 9.2, pp. 137–164. URL: http://dl.acm.org/citation.cfm?id=356692.

Sennrich, Rico, Barry Haddow, and Alexandra Birch (2016). “Improving Neural Machine Translation Models with Monolingual Data”. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, pp. 86–96. DOI: 10.18653/v1/P16-1009.

Serban, Iulian Vlad, Alberto García-Durán, Caglar Gulcehre, Sungjin Ahn, Sarath Chandar, Aaron Courville, and Yoshua Bengio (2016). “Generating Factoid Questions With Recurrent Neural Networks: The 30M Factoid Question-Answer Corpus”. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, pp. 588–598. DOI: 10.18653/v1/P16-1056. URL: http://aclweb.org/anthology/P16-1056.

Sgall, Petr, Eva Hajicová, Eva Hajicová, Jarmila Panevová, and Jarmila Panevova (1986). *The meaning of the sentence in its semantic and pragmatic aspects*. Springer Science & Business Media.

Shi, Tianze, Felix G. Wu, Xilun Chen, and Yao Cheng (2017). “Combining Global Models for Parsing Universal Dependencies”. In: *Proceedings of the CoNLL*

2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies. Vancouver, Canada: Association for Computational Linguistics, pp. 31–39. DOI: 10.18653/v1/K17-3003. URL: https://www.aclweb.org/anthology/K17-3003.

Simard, Patrice Y., Dave Steinkraus, and John C. Platt (2003). "Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis". In: *Proceedings of the 7th International Conference on Document Analysis and Recognition*, pp. 958–. URL: http://research.microsoft.com/apps/pubs/default.aspx?id=68920.

Smith, Aaron, Bernd Bohnet, Miryam de Lhoneux, Joakim Nivre, Yan Shao, and Sara Stymne (2018). "82 Treebanks, 34 Models: Universal Dependency Parsing with Multi-Treebank Models". In: *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*. Brussels, Belgium: Association for Computational Linguistics, pp. 113–123. DOI: 10.18653/v1/K18-2011. URL: https://www.aclweb.org/anthology/K18-2011.

Smith, Carl H (1982). "The power of pluralism for automatic program synthesis". In: *Journal of the ACM (JACM)* 29.4, pp. 1144–1165.

Smith, David A. and Jason Eisner (2009). "Parser Adaptation and Projection with Quasi-Synchronous Grammar Features". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pp. 822–831. URL: http://www.aclweb.org/anthology/D/D09/D09-1086.

Smith, Noah A. (2006). "Novel Estimation Methods for Unsupervised Discovery of Latent Structure in Natural Language Text". PhD thesis. Baltimore, MD: Johns Hopkins University. URL: http://cs.jhu.edu/~jason/papers/#smith-2006.

Smith, Noah A. and Jason Eisner (2005). "Guiding Unsupervised Grammar Induction Using Contrastive Estimation". In: *International Joint Conference on Artificial Intelligence (IJCAI) Workshop on Grammatical Inference Applications*. Edinburgh, pp. 73–82. URL: http://cs.jhu.edu/~jason/papers/#smith-eisner-2005-gia.

Smith, Noah A. and Jason Eisner (2006). "Annealing Structural Bias in Multilingual Weighted Grammar Induction". In: *Proceedings of the International Conference on Computational Linguistics and the Association for Computational Linguistics (COLING-ACL)*. Sydney, pp. 569–576. URL: http://cs.jhu.edu/~jason/papers/#smith-eisner-2006-acl-sa.

Smola, Alex J and Bernhard Schölkopf (2004). "A tutorial on support vector regression". In: *Statistics and computing* 14.3, pp. 199–222.

Socher, Richard, Christopher D Manning, and Andrew Y Ng (2010). "Learning continuous phrase representations and syntactic parsing with recursive neural networks". In: *Proceedings of the NIPS-2010 Deep Learning and Unsupervised Feature Learning Workshop*. Vol. 2010, pp. 1–9.

Socher, Richard, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts (2013). "Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank". In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Seattle, Washington, USA: Association for Computational Linguistics, pp. 1631–1642. URL: https://www.aclweb.org/anthology/D13-1170.

Song, Jae Jung (2014). *Linguistic Typology: Morphology and Syntax*. Routledge. URL: https://books.google.com/books/about/Linguistic_typology.html?id=TiULAQAAMAAJ.

Spitkovsky, Valentin I. (2013). "Grammar Induction and Parsing with Dependency-and-Boundary Models". PhD thesis. Stanford, CA: Computer Science Department, Stanford University. URL: http://nlp.stanford.edu/pubs/SpitkovskyThesis.pdf.

Spitkovsky, Valentin I., Hiyan Alshawi, and Daniel Jurafsky (2010). "From Baby Steps to Leapfrog: How "Less is More" in Unsupervised Dependency Parsing". In: *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Los Angeles, California, pp. 751–759. URL: http://aclweb.org/anthology/N10-1116.

Spitkovsky, Valentin I., Hiyan Alshawi, and Daniel Jurafsky (2012). "Three Dependency-and-Boundary Models for Grammar Induction". In: *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Jeju Island, Korea: Association for Computational Linguistics, pp. 688–698. URL: https://www.aclweb.org/anthology/D12-1063.

Spitkovsky, Valentin I., Hiyan Alshawi, and Daniel Jurafsky (2013). "Breaking Out of Local Optima with Count Transforms and Model Recombination: A Study in Grammar Induction". In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Seattle, Washington, USA, pp. 1983–1995. URL: http://aclweb.org/anthology/D13-1204.

Strubell, Emma, Patrick Verga, Daniel Andor, David Weiss, and Andrew McCallum (2018). "Linguistically-Informed Self-Attention for Semantic Role Labeling". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, pp. 5027–5038. URL: https://www.aclweb.org/anthology/D18-1548.

Stuhlmüller, Andreas, Jacob Taylor, and Noah Goodman (2013). “Learning stochastic inverses”. In: *Advances in neural information processing systems*, pp. 3048–3056.

Sutskever, Ilya, Oriol Vinyals, and Quoc V Le (2014). “Sequence to Sequence Learning with Neural Networks”. In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger. Curran Associates, Inc., pp. 3104–3112. URL: http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf.

Täckström, Oscar, Ryan McDonald, and Joakim Nivre (2013). “Target Language Adaptation of Discriminative Transfer Parsers”. In: *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Atlanta, Georgia, pp. 1061–1071. URL: http://aclweb.org/anthology/N13-1126.

Täckström, Oscar, Dipanjan Das, Slav Petrov, Ryan McDonald, and Joakim Nivre (2013). “Token and Type Constraints for Cross-Lingual Part-of-Speech Tagging”. In: *Transactions of the Association for Computational Linguistics* 1, pp. 1–12. URL: http://aclweb.org/anthology/Q13-1001.

Taskar, Ben, Dan Klein, Michael Collins, Daphne Koller, and Christopher Manning (2004). “Max-Margin Parsing”. In: *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*. URL: http://www.aclweb.org/anthology/W04-3201.

Taskar, Ben, Vassil Chatalbashev, Daphne Koller, and Carlos Guestrin (2005). “Learning Structured Prediction Models: A Large Margin Approach”. In: *Proceedings of the 22nd International Conference on Machine Learning*, pp. 896–903. ISBN: 1-59593-180-5. URL: http://doi.acm.org/10.1145/1102351.1102464.

Tesnière, Lucien (1959). *Eléments de syntaxe structurale*. Klincksieck.

Tiedemann, Jörg (2014). “Rediscovering Annotation Projection for Cross-Lingual Parser Induction”. In: *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pp. 1854–1864. URL: http://www.aclweb.org/anthology/C14-1175.

Tiedemann, Jörg and Zeljko Agić (2016). “Synthetic treebanking for cross-lingual dependency parsing”. In: *Journal of Artificial Intelligence Research* 55, pp. 209–248.

Tiedemann, Jörg, Željko Agić, and Joakim Nivre (2014). “Treebank Translation for Cross-Lingual Parser Induction”. In: *Proceedings of the Eighteenth Conference* *on Computational Natural Language Learning*. Ann Arbor, Michigan, pp. 130–140. DOI: 10.3115/v1/W14-1614. URL: http://www.aclweb.org/anthology/W14-1614.

Tieleman, Tijmen and Geoffrey Hinton (2012). *Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude*. COURSERA: *Neural Networks for Machine Learning*. URL: https://www.coursera.org/learn/neural-networks/lecture/YQHki/rmsprop-divide-the-gradient-by-a-running-average-of-its-recent-magnitude.

Tu, Kewei (2015). "Stochastic And-Or grammars: A unified framework and logic perspective". In: *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2016)*.

Vapnik, Vladimir (2013). *The nature of statistical learning theory*. Springer science & business media.

Varjokallio, Matti and Dietrich Klakow (2016). "Unsupervised morph segmentation and statistical language models for vocabulary expansion". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Berlin, Germany, pp. 175–180. URL: http://aclweb.org/anthology/P16-2029.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin (2017). "Attention is All you Need". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., pp. 5998–6008. URL: http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf.

Vilares, Jesús, Manuel Vilares, and Juan Otero (2011). "Managing Misspelled Queries in IR Applications". In: *Information Processing & Management* 47.2, pp. 263–286. URL: http://dl.acm.org/citation.cfm?id=1945179.

Wang, Dingquan and Jason Eisner (2016). "The Galactic Dependencies Treebanks: Getting More Data by Synthesizing New Languages". In: *Transactions of the Association of Computational Linguistics* 4, pp. 491–505. URL: http://cs.jhu.edu/~jason/papers/#wang-eisner-2016.

Wang, Dingquan and Jason Eisner (2017). "Fine-Grained Prediction of Syntactic Typology: Discovering Latent Structure with *Supervised* Learning". In: *Transactions of the Association for Computational Linguistics* 5, pp. 147–161. URL: https://transacl.org/ojs/index.php/tacl/article/view/1060.

Wang, Dingquan and Jason Eisner (2018a). "Surface Statistics of an Unknown Language Indicate How to Parse It". In: *Transactions of the Association for Computational Linguistics* 6, pp. 667–685. DOI: 10.1162/tacl_a_00248. URL: https://www.aclweb.org/anthology/Q18-1046.

Wang, Dingquan and Jason Eisner (2018b). "Synthetic Data Made to Order: The Case of Parsing". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Brussels, pp. 1325–1337. URL: https://www.cs.jhu.edu/~jason/papers/#wang-eisner-2018-emnlp.

Wang, Qin Iris, Dale Schuurmans, and Dekang Lin (2008). "Semi-Supervised Convex Training for Dependency Parsing". In: *Proceedings of the 2008 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Columbus, Ohio: Association for Computational Linguistics, pp. 532–540. URL: https://www.aclweb.org/anthology/P08-1061.

Wang, Yushi, Jonathan Berant, and Percy Liang (2015a). "Building a Semantic Parser Overnight". 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)*. Beijing, China: Association for Computational Linguistics, pp. 1332–1342. DOI: 10.3115/v1/P15-1129.

Wang, Yushi, Jonathan Berant, and Percy Liang (2015b). "Building a Semantic Parser Overnight". 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)*. Beijing, China, pp. 1332–1342. URL: http://aclweb.org/anthology/P15-1129.

Weston, Jason, Antoine Bordes, Sumit Chopra, and Tomas Mikolov (2016). "Towards AI-Complete Question Answering: A Set of Prerequisite Toy Tasks". In: *Proceedings of the International Conference on Learning Representations*. URL: http://arxiv.org/abs/1502.05698.

Xu, Keyulu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka (2019). "How Powerful are Graph Neural Networks?" In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=ryGs6iA5Km.

Yamada, Kenji and Kevin Knight (2001). "A Syntax-based Statistical Translation Model". In: *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*. Toulouse, France: Association for Computational Linguistics, pp. 523–530. DOI: 10.3115/1073012.1073079.

Yarowsky, David, Grace Ngai, and Richard Wicentowski (2001). "Inducing Multilingual Text Analysis Tools via Robust Projection across Aligned Corpora". In:

Proceedings of the First International Conference on Human Language Technology Research. URL: http://www.aclweb.org/anthology/H01-1035.

Yuret, Deniz (1998). "Discovery of linguistic relations using lexical attraction". In: *arXiv preprint cmp-lg/9805009*.

Zeman, Daniel, Martin Popel, Milan Straka, Jan Hajič, Joakim Nivre, Filip Ginter, Juhani Luotolahti, Sampo Pyysalo, Slav Petrov, Martin Potthast, Francis Tyers, Elena Badmaeva, Memduh Gokirmak, Anna Nedoluzhko, Silvie Cinková, Jan Hajič jr., Jaroslava Hlaváčová, Václava Kettnerová, Zdeňka Urešová, Jenna Kanerva, Stina Ojala, Anna Missilä, Christopher D. Manning, Sebastian Schuster, Siva Reddy, Dima Taji, Nizar Habash, Herman Leung, Marie-Catherine de Marneffe, Manuela Sanguinetti, Maria Simi, Hiroshi Kanayama, Valeria de Paiva, Kira Droganova, Héctor Martínez Alonso, Çağrı Çöltekin, Umut Sulubacak, Hans Uszkoreit, Vivien Macketanz, Aljoscha Burchardt, Kim Harris, Katrin Marheinecke, Georg Rehm, Tolga Kayadelen, Mohammed Attia, Ali Elkahky, Zhuoran Yu, Emily Pitler, Saran Lertpradit, Michael Mandl, Jesse Kirchner, Hector Fernandez Alcalde, Jana Strnadová, Esha Banerjee, Ruli Manurung, Antonio Stella, Atsuko Shimada, Sookyoung Kwak, Gustavo Mendonça, Tatiana Lando, Rattima Nitisaroj, and Josie Li (2017). "CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies". In: *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*. Vancouver, Canada: Association for Computational Linguistics, pp. 1–19. DOI: 10.18653/v1/K17-3001. URL: https://www.aclweb.org/anthology/K17-3001.

Zeman, Daniel, Jan Hajič, Martin Popel, Martin Potthast, Milan Straka, Filip Ginter, Joakim Nivre, and Slav Petrov (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*. Brussels, Belgium: Association for Computational Linguistics, pp. 1–21. DOI: 10.18653/v1/K18-2001. URL: https://www.aclweb.org/anthology/K18-2001.

Zettlemoyer, Luke and Michael Collins (2007). "Online Learning of Relaxed CCG Grammars for Parsing to Logical Form". In: *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*. Prague, Czech Republic: Association for Computational Linguistics, pp. 678–687. URL: https://www.aclweb.org/anthology/D07-1071.

Zhang, Yuan and Regina Barzilay (2015). "Hierarchical Low-Rank Tensors for Multilingual Transfer Parsing". In: *Proceedings of the 2015 Conference on Empirical*

Methods in Natural Language Processing. Lisbon, Portugal: Association for Computational Linguistics, pp. 1857–1867. DOI: 10.18653/v1/D15-1213. URL: https://www.aclweb.org/anthology/D15-1213.

Zhang, Yuan, Roi Reichart, Regina Barzilay, and Amir Globerson (2012). "Learning to Map into a Universal POS Tagset". In: *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Jeju Island, Korea, pp. 1368–1378. URL: http://aclweb.org/anthology/D12-1125.

Zhang, Yuan, David Gaddy, Regina Barzilay, and Tommi Jaakkola (2016). "Ten Pairs to Tag—Multilingual POS Tagging via Coarse Mapping between Embeddings". In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. San Diego, California, pp. 1307–1317. DOI: 10.18653/v1/N16-1156. URL: http://aclweb.org/anthology/N16-1156.

Zhu, Song-Chun, David Mumford, et al. (2007). "A stochastic grammar of images". In: *Foundations and Trends® in Computer Graphics and Vision* 2.4, pp. 259–362.

Zmigrod, Ran, Sebastian J. Mielke, Hanna Wallach, and Ryan Cotterell (2019). "Counterfactual Data Augmentation for Mitigating Gender Stereotypes in Languages with Rich Morphology". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, pp. 1651–1661. DOI: 10.18653/v1/P19-1161.

Zwicky, Arnold M (1985). "Heads". In: *Journal of linguistics* 21.1, pp. 1–29.

Wang, Dingquan

CONTACT INFORMATION

Hackerman 321, 3400 N. Charles Street, Baltimore, MD, 21218
Site: www.cs.jhu.edu/~wdd

Mobile: +1 (917)680-9648
E-mail: wdd@cs.jhu.edu
GitHub: github.com/wddabc

RESEARCH INTERESTS

Natural language processing and machine learning, with a focus on syntactic analysis on low-resource langauges via un/semi-supervised learning methods.

EDUCATION

Johns Hopkins University, Baltimore, US
Whiting School of Engineering
Doctor of Philosophy, Computer Science, September, 2014-

- Adviser: Jason Eisner

Columbia University, New York, US
The Fu Foundation School of Engineering & Applied Science
Masters, Computer Science, September, 2012- Decemeber, 2013, GPA: 3.95/4.0

- Advisers: Rebecca J. Passonneau and Michael Collins

Shanghai Jiao Tong University, Shanghai, China
Computer Science and Technology(ACM Honored Class)
B.S., Engineering, July, 2011, Major GPA: 3.74/4.3

- Dissertation Topic: "Intent Based Query Clustering on User Logs"
- Adviser: Yong Yu

PUBLICATIONS

Xiang Lisa Li*, **Dingquan Wang*** and Jason Eisner: A Generative Model for Punctuation in Dependency Trees, *Transactions of the Association for Computational Linguistics (TACL)*, 2019, to appear

Dingquan Wang and Jason Eisner: Surface statistics of an unknown language indicate how to parse it, *Transactions of the Association for Computational Linguistics (TACL)*, 2019, to appear

Dingquan Wang and Jason Eisner: Synthetic data made to order: The case of parsing, In *2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Brussels, Belgium, 2018, oral presentation

Dingquan Wang and Jason Eisner: Predicting fine-grained syntactic typology from surface features, *Society for Computation in Linguistics (SCiL)*, Salt Lake City, US, 2018

Dingquan Wang, Nanyun Peng and Kevin Duh: A Multi-task Learning Approach to Adapting Bilingual Word Embeddings for Cross-lingual Named Entity Recognition, In *Proceedings of the International Joint Conference on Natural Language Processing (IJCNLP)*, Taipei, Taiwan, 2017, oral presentation

Dingquan Wang and Jason Eisner: Fine-grained prediction of syntactic typology: Discovering latent structure with supervised learning, *Transactions of the Association for Computational Linguistics (TACL)*, 2017, oral presentation at ACL 2017

Dingquan Wang and Jason Eisner: The Galactic Dependencies treebanks: Getting more data by synthesizing new languages, *Transactions of the Association for Computational Linguistics (TACL)*, 2016, oral presentation at EMNLP 2016

Ruihua Song, **Dingquan Wang**, Jian-Yun Nie, Ji-Rong Wen and Yong Yu: Enhancing Web Search with Queries of Equivalent Intents, *Information Retrieval Journal*, 2016

Dingquan Wang, Rebecca J. Passonneau, Michael Collins and Cynthia Rudin: Modeling Weather Impact on a Secondary Electrical Grid, In *Proceedings of the the 4th International Conference on Sustainable Energy Information Technology (SEIT-2014)*

Boyi Xie, **Dingquan Wang** and Rebecca J. Passonneau: Semantic Feature Representation to Capture News Impact, In *Proceedings of the 27th International Florida Artificial Intelligence Research Society Conference, FLAIRS 2014*

Dingquan Wang, Weinan Zhang, Gui-Rong Xue and Yong Yu: Deep Classifier for Large Scale Hierarchical Text Classification, In *the 1st PASCAL Challenge on Large Scale Hierarchical Text Classification*

Weinan Zhang, **Dingquan Wang**, Gui-Rong Xue and Hongyuan Zha: Advertising Keywords Recommendation for Short-text Web Pages using Wikipedia, *ACM Transactions on Intelligent Systems and Technology* Vol. 3, No. 2. DOI=10.1145/2089094.2089112

Work Experience

Microsoft Research Summer, 2015
Machine Learning Department Mentor: Matthew Richardson and Scott Yih
Research internship
Neural network models for machine reading comprehension

Mobvoi Inc. Summer, 2014
Natural Language processing Group Mentor: Libin Shen
Research internship
Unsupervised method for generating Chinese abbreviations

Microsoft Research Asia July, 2011 - September, 2011 and July, 2010 - February, 2011
Web Data Management Group Mentor: Ruihua Song
Research Internship
Query clustering on user logs

Teaching Experience

Johns Hopkins University

Artificial Intelligence	Benjamin Van Durme, Fall, 2018
Artificial Intelligence	Benjamin Van Durme, Spring, 2018
Natural Language Processing	Jason Eisner, Spring, 2016
Natural Language Processing	Jason Eisner, Fall, 2014

Columbia University

Advanced Machine Learning	Tony Jebara, Spring, 2013
Machine Learning	Tony Jebara, Fall, 2012

Shanghai Jiao Tong University

Mathematics in Computer Science	John E. Hopcroft, Spring, 2012
Computer Organization Lab	ACM Honored Class, Spring, 2010

Honors and Awards

MSTA Fellowship of Columbia University	Spring, 2013
Microsoft Excellent Internship Award	April,2011
Microsoft Young Fellowship Award,	May, 2010
Excellent Academic Scholarship, Shanghai Jiao Tong University	2007–2010

Professional Services

Reviewer: ACL(2017), EMNLP(2017,2018), NAACL(2019), CCL(2017), IJCNLP(2017), AAAI(2018)

COMPUTER SKILLS	**OS, Tools and IDEs**: MacOS, Linux, Windows, Latex, Pytorch, Theano, Keras, Fabric, Makefile, SGE, Slurm, Gnuplot, Vim, IntelliJ IDEA **Programming Languages**: Python, Java, C, C++, C#, Matlab, Perl, R