Incremental Generative Models for Syntactic and Semantic Natural Language Processing

Jan Moolman Buys

University College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Trinity 2017

Aan my ouers

Acknowledgements

Many thanks go to my supervisor, Phil Blunsom, for his advice and support throughout this study. I thank him for giving me the freedom to pursue problems I found interesting and helping me to navigate the fast-changing landscape of deep learning and NLP. The depth and breadth of his knowledge about NLP and machine learning never ceased to amaze me, and it is hard to fully appreciate the value of everything I learned from him. Thanks to my fellow students in the Oxford Computational Linguistics group for their camaraderie in the past few years, especially Yishu Miao and Lei Yu who shared most of the journey with me.

I thank the Clarendon Fund and the Skye Foundation for funding this study, as well as University College for travel grants. Thanks to Julie Sheppard for helping to make dealing with Oxford bureaucracy relatively painless. Jan Botha hosted me for an internship at Google London - thanks for making this a pleasant summer and a productive experience.

Thanks to Stephan Oepen for valuable assistance with the English Resource Grammar and Minimal Recursion Semantics datasets, and to him and Emily Bender for valuable discussions on the broader context of MRS parsing. Thanks also to Chris Dyer for valuable feedback.

Thanks to my housemates over the years on Merton Street and Abingdon Road for friendship and providing distractions from study. The broader graduate community at University College was very supportive in helping me adapt in a new country and to Oxford. I was fortunate to meet many amazing people through the community of Clarendon Scholars. Thanks to the members of the Univ chapel choir, Oxford University Wind Orchestra and the Oxford Bach Choir for memorable musical experiences. Thanks to friends and family from South Africa for keeping in touch and visiting.

Final gratitude goes to my parents for their continued support throughout my education, without which I would not have been able to come this far.

Abstract

This thesis investigates the role of linguistically-motivated generative models of syntax and semantic structure in natural language processing (NLP). Syntactic well-formedness is crucial in language generation, but most statistical models do not account for the hierarchical structure of sentences. Many applications exhibiting natural language understanding rely on structured semantic representations to enable querying, inference and reasoning. Yet most semantic parsers produce domain-specific or inadequately expressive representations.

We propose a series of generative transition-based models for dependency syntax which can be applied as both parsers and language models while being amenable to supervised or unsupervised learning. Two models are based on Markov assumptions commonly made in NLP: The first is a Bayesian model with hierarchical smoothing, the second is parameterised by feed-forward neural networks. The Bayesian model enables careful analysis of the structure of the conditioning contexts required for generative parsers, but the neural network is more accurate. As a language model the syntactic neural model outperforms both the Bayesian model and n-gram neural networks, pointing to the complementary nature of distributed and structured representations for syntactic prediction. We propose approximate inference methods based on particle filtering. The third model is parameterised by recurrent neural networks (RNNs), dropping the Markov assumptions. Exact inference with dynamic programming is made tractable here by simplifying the structure of the conditioning contexts.

We then shift the focus to semantics and propose models for parsing sentences to labelled semantic graphs. We introduce a transition-based parser which incrementally predicts graph nodes (predicates) and edges (arguments). This approach is contrasted against predicting top-down graph traversals. RNNs and pointer networks are key components in approaching graph parsing as an incremental prediction problem. The RNN architecture is augmented to condition the model explicitly on the transition system configuration. We develop a robust parser for Minimal Recursion Semantics, a linguistically-expressive framework for compositional semantics which has previously been parsed only with grammar-based approaches. Our parser is much faster than the grammar-based model, while the same approach improves the accuracy of neural Abstract Meaning Representation parsing.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Natural language enables humans to express complex thoughts through linear sequences of discrete symbols (words), usually organised into sentences. The *syntax* of natural language defines which words and sentences are well-formed. Although language is expressed as strings, its syntactic structure is hierarchical (Chomsky, 1957; Everaert et al., 2015) which linguists typically characterise as trees. Syntax also drives the *semantics* of natural language as the meaning of words and phrases are composed into larger units, enabling the expression of complex relations between entities, eventualities and modifiers, typically expressed by directed graphs.

There has recently been a surge in the use and usability of applications that are driven by natural language processing (NLP). This has been driven by the coalescence of the availability of unprecedented amounts of natural language data, on-device and cloud-based computational power, and advances in machine learning driven by neural networks. *Deep learning* refers to the process of training large, multilayer neural networks on large amounts of data, and their application to various machine learning problems. Example NLP applications include machine translation, automatic speech recognition and synthesis, dialogue-based personal and home assistants, text prediction, and automatic extraction of structured information from text. Language processing is also employed for web-search, automatic advertisement placement and stock market prediction, and has huge potential in the biomedical and legal domains.

Traditionally, many NLP models for language understanding or generation tasks are pipeline approaches which perform linguistic analysis at different levels (including syntax and semantics) and then make predictions based on sparse features over the predicted linguistic representations. In contrast neural networks have been shown to be very successful at learning vector embeddings of words that capture lexical semantics, as well as embedding phrases or sentences and representing words in context. Many neural network NLP models are trained *end-to-end*; all the components of the model are optimised together from the raw input with an objective function based on the task, rather than learning to predict linguistic structure as an intermediate step. Deep learning models with relatively simple underlying structure and little feature engineering (though containing millions of learnable parameters and requiring careful selection of hyperparameters that control model capacity and learning) have improved the state of the art in a range of NLP tasks. This raises questions about the future role of explicit models of syntactic and semantic structure in natural language processing.

In this thesis we argue that structured and distributed representations are complementary in their representational and predictive power for natural language processing. Neural networks can be used to predict more expressive structured representations more accurately than previously possible. This creates more opportunities for using structured representations in downstream NLP applications. On the other hand, neural models trained for structured prediction can learn more informative distributed representations, which can be used in end-to-end models without necessarily preforming inference over the structure during application.

We propose *generative* and *incremental* models of syntactic and semantic structure. We define probabilistic models over sentences, syntax trees and semantic graphs. We view syntax both in terms of predicting dependency trees, and part of generative models that define probability distributions over sentences. We also propose probabilistic models for linguistically-expressive semantic graphs that are aligned to words and phrases in the sentences they represent and grounded in broad-coverage lexicons.

1.1 Background

1.1.1 Syntactic and semantic structure

The structure of language is deeper than the observed sequential structure. In order to recognize whether sentences are well-formed, and to assign meaning to them, their syn-

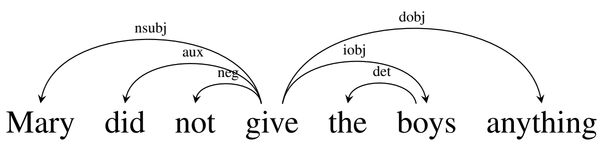


Figure 1.1: A syntactic dependency tree. Arcs indicate *head* → *modifier* dependencies, and are labelled with dependency relations.

tactic structure has to be modelled. To illustrate the need for syntax to determine well-formedness, we give two examples (* indicates ill-formedness):

1. (a) Mary did not give the boys anything.

   (b) Mary gave the boys nothing.

   (c) * Mary gave the boys anything.

2. (a) The boys who went to school are still away.

   (b) * The boys who went to school is still away.

The first example concerns negative polarity items, which require an overt negative element (Everaert et al., 2015). Negation can be expressed either by negating the verb (*did not give*) or the polarity item (*nothing*), but a positive statement would require a different construction. The relation between the two positions is determined by the syntactic structure of the sentence, not the linear ordering (there can be an arbitrary long distance between them). The second example displays subject-verb agreement in English, where the number of the subject (singular or plural) determines the form of the verb (*is/are*). Again, the subject-verb relation is determined by hierarchical rather than linear structure.

Syntactic structure is usually expressed as trees. In this thesis we focus on dependency trees (see Section 2.2 for definitions), which denote syntactic structure through a tree structure in which the words in the sentence are the nodes in a graph and the edges between nodes represent relations between words (known as *bilexical* relations). Dependency parsing is widely used in NLP due to being light-weight relative to constituency parsing or more linguistically expressive formalisms, while dependency relations are equivalent or close to semantic relations which can be used for downstream tasks. Dependency treebanks are available in over 40 languages.[1] An example dependency tree is given in Figure 1.1.

[1]http://universaldependencies.org

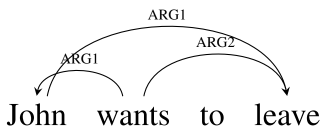


Figure 1.2: A semantic dependency graph.

While syntax plays an important role in determining semantic interpretation in most linguistic theories, syntactic structure is not equivalent to semantic structure. In syntactic dependencies nodes can only have one head (forming a tree structure), while in semantic dependencies that is not the case. Furthermore, not all the words in a sentence have to participate in semantic relations, while there might also be implicit semantic concepts which play a role in determining the meaning but do not correspond to the surface form of any words. Figure 1.2 gives an example bilexical semantic graph (more expressive semantic representations are introduced in Chapter 5). In the example *John* is a semantic argument of both the predicates *wants* and *leave*.

1.1.2 Generative models

A generative model defines a joint probability distribution $p(\mathbf{x}, \mathbf{y})$ over the observed data $\mathbf{y}$ and the output or structure to be predicted $\mathbf{x}$. In contrast discriminative models define a conditional distribution $p(\mathbf{x}|\mathbf{y})$. In this thesis we propose generative syntactic models that define joint probability distributions over sentences (word sequences) and their dependency parses. Additionally we propose conditional models for semantic structure that model a joint distribution over semantic concepts and the relations between them, conditioned on the input sentence.

There are a number of motivations for focussing on generative models:

- They model the observed data by generating, or assigning probabilities to strings. Therefore generative models enable us to formulate and test hypotheses about the underlying structure or processes which generate the data, rather than modelling the structure in isolation.

- They are probabilistic models that provide estimates for the uncertainty in predictions. While some discriminative models are probabilistic, many are not. This interpretability is crucial for many real-world applications.

- The structure that we want to predict can be treated as latent variables over which inference can be performed during training or decoding. This enables us to perform semi-supervised or unsupervised learning, and to estimate the marginal distribution over the observed data.

- Locally-normalised discriminative models, where the conditional distribution factors into the product of the probabilities of the elements in the predicted structure, suffer from the label bias problem (Lafferty et al., 2001). This refers to the phenomenon that models can learn to ignore parts of the input that are not required to make locally good decisions, and that it might be overconfident in local decisions which might not be globally optimal. Globally normalised discriminative models do not suffer from this problem, but training is more expensive as inference has to be performed over the output space during training. Explaining-away effects have also been manifested in discriminative neural sequence-to-sequence models (Sutskever et al., 2014; Bahdanau et al., 2015) that often ignore parts of the input sequence by not predicting any output corresponding to it during decoding.

- It has been shown both theoretically and practically that generative models have lower sample complexity than discriminative models (Ng and Jordan, 2002; Yogatama et al., 2017): Discriminative models are usually more accurate given sufficient training data, but generative models reach their asymptotic error rates faster, meaning that when small amounts of training data are available generative models will often be more accurate than their discriminative counterparts.

One of the limitations of generative models is that they often have to make strong independence assumptions in order to make inference tractable. Additionally many probabilistic density estimators in generative models are not sufficiently powerful to model complex data distributions accurately. Recent advances in neural networks enable us to overcome these disadvantages. Neural networks have been shown to learn good representations of input contexts that can capture long-distance dependencies. They can also approximate arbitrary non-linear functions, which enables them to be good probability estimators. In computer vision neural networks have been shown to be successful generative models both quantitatively and qualitatively (Goodfellow et al., 2014; van den Oord et al., 2016).

1.1.3 Linguistic structure prediction

Two broad areas for the application of NLP models are language generation and language understanding tasks. In this thesis syntactic structure is used primarily in the context of language generation, specifically language modelling, while semantic structure is predicted with the goal of contributing to language understanding tasks. Syntactic structure could still have a role in language understanding tasks, but as the information required is primarily semantic, we focus on predicting semantic representations directly. Language generation can also be performed from semantic structures, but we do not address that task here.

Language modelling Language modelling, the task of estimating probability distributions over word sequences, is a crucial component of many language generation applications. Examples include automatic speech recognition, machine translation, text prediction on mobile keyboards, automatic e-mail reply suggestions and dialogue systems. Further language generation tasks that recently drew attention include image caption generation (Vinyals et al., 2015c) and recipe generation (Kiddon et al., 2016).

Most language models do not explicitly account for the hierarchical structure of language. One reason is that syntactic language models are considerably slower, as they have to do inference over structure during decoding; another reason is that in practice language models based on sequential structure can estimate the next word distribution very well and can be trained on very large corpora. However, syntactic structure has been shown to improve the intrinsic and extrinsic performance of language models (Chelba and Jelinek, 2000; Emami and Jelinek, 2005). It could be particularly beneficial in languages and domains that are not well enough resourced for learning accurate language models.

Incremental parsing In this thesis *incrementally* refers to two related concepts: Linear-time processing, and online prediction (predicting partial output after observing partial input). In incremental models the output is predicted as a sequence. Incrementality is essential in generative syntactic models, as one of the defining characteristics of a language model is the ability to predict the distribution over the next word in a sequence given past words. In a generative incremental parser the goal is to be able to do online processing to estimate the distribution over the next word in the sentence by marginalising (approximately or exactly) over the syntactic structure of the partially observed sentence.

Transition-based dependency parsers (Nivre, 2008) are a class of incremental parsers that offer an appealing trade-off between speed and accuracy, often matching or exceeding the accuracy of graph-based parsers that use dynamic programming to perform inference over the entire search space (McDonald et al., 2005b; Koo et al., 2010; Lei et al., 2014). Parsing is formulated as executing a sequence of shift-reduce actions on a stack and a buffer. Decoding is performed either greedily or with beam-search. We shall focus on generative transition-based models that jointly predict the observed words and dependency parses.

The principles of incrementality can also be applied to semantic graph parsing. Here the goal is to incrementally predict the nodes and edges of semantic graphs, as exact inference is intractable unless strong assumptions are made about the class of graph structures to be predicted and the relation between the input tokens and graph nodes. Transition-based parsing, generative in the sense that the graph nodes and edges are predicted jointly, is an appropriate abstract computational system for modelling this task.

1.2 Aims

Two important goals in Natural Language Processing (NLP) are the development of parsers that can produce linguistic analyses of sentences with high speed and accuracy, and of models that are able to generate well-formed sentences. We formulate the following aims in order to advance these goals:

- Develop fast and accurate generative syntactic dependency models for parsing and language modelling that enable approximate or exact online inference.
- Develop a robust, fast and accurate parser for linguistically-expressive semantic graphs.

Our thesis is that distributed representations (which are learned by neural networks) and linguistically-motivated structured representations are complementary in models that form the basis of language generation and understanding applications. More concretely this thesis is based on the following hypotheses:

- Neural networks are more accurate than count-based estimators for both language modelling and parsing.
- Generative syntactic models improve the intrinsic performance of language modelling over purely sequential models for both count-based and neural approaches.
- Neural models can be applied to predict more complex linguistically-motivated structures than previously feasible with data-driven approaches. In particular they can predict rich semantic representations without first deriving the syntax.
- Modelling structure in the architecture of neural incremental parsing models improves performance over treating it as a purely sequential prediction problem.

1.3 Contributions

The main contributions made by this thesis are as follows:

- We proposed (with the publication of Buys and Blunsom (2015b) and Buys and Blunsom (2015a)) the first generative models that can be employed as both accurate dependency parsers and language models.
- We propose an inference algorithm for generative dependency parsing based on particle filtering that offers a better trade-off between speed and accuracy than previous approaches.
- We propose a Bayesian model for generative dependency parsing with sophisticated smoothing over carefully chosen priors over ordered contexts. A fine-grained analysis of the structure of the conditioning contexts shows that our Bayesian model is more accurate than discriminative models with access to the same context.
- We show that generative parsers based on feed-forward neural networks are more accurate than the Bayesian models and competitive with the accuracy of the best previous generative parser, while being much faster.
- We propose a generative parser based on recurrent neural networks (RNNs) in which exact inference with dynamic programming is tractable. The RNN enables the model

to be accurate using a very small feature set based on the representations produced by the RNN, which enables us to apply a dynamic program to the model. This model is a more accurate parser than our feed-forward models, but failed to improve perplexity over an RNN language model.

- We show that it is feasible to perform unsupervised learning of syntactic structure with the RNN generative model when optimising the language modelling objective by backpropagating through the dynamic program. The model can learn non-trivial trees with plausible hierarchical structure.

- We propose the first *robust* and *scalable* parser for Minimal Recursion Semantics, a semantic representation that has previously been predicted only in the context of grammar-based parsing, which has high precision but incomplete coverage.

- We propose a transition-based neural semantic graph parser that can predict unrestricted semantic graphs, while previous approaches have been constrained to specific formalisms or graph classes. Additionally we show that pointer networks can be used to predict the spans of the alignments between graph nodes and input tokens.

- We show that a simple, well-tuned model for neural Abstract Meaning Representation (AMR) parsing outperforms previous neural approaches, while the transition-based parser together with heuristics to augment the structure of AMR graphs results in better performance than other approaches using comparable resources.

1.4 Thesis outline

Chapter 2: Background: Language modelling and parsing

We review different classes of language models, which form the basis of the generative models we develop in the thesis: Bayesian hierarchical Pitman-Yor processes (HPYPs), feed-forward neural networks and recurrent neural networks. After reviewing syntactic and semantic representations, we define the transition systems for dependency parsing that are used in this thesis. Finally we review approaches to syntactic dependency parsing.

Chapter 3: Markov Models for Generative Dependency Parsing

We propose two models for generative dependency parsing based on models making Markov assumptions. The first is a Bayesian model, the second a feed-forward neural network. We introduce decoding and inference algorithms based on particle-filtering. The Bayesian model obtains an unlabelled attachment accuracy of 88.47% while the neural network improves this to 90.16%. While these models are less accurate than state-of-the-art discriminative parsers they are more accurate than most previous generative parsers and less sophisticated discriminative models. They are also faster than previous generative parsers during both training and decoding. For language modelling both generative parsers outperform their n-gram counterparts. We show that fine-tuning the model trained with annotated parse trees to optimise the marginal likelihood further improves language modelling quality, giving a test set perplexity of 138.6 for the neural model, compared to 170.1 for the count-based n-gram baseline. A qualitative evaluation shows that the model is able to generate sentences that display both local and syntactic coherence. This chapter is based on research published as Buys and Blunsom (2015b) and Buys and Blunsom (2015a).

Chapter 4: Recurrent Models for Generative Dependency Parsing

We introduce an RNN model for generative transition-based parsing. The RNN enables us to restrict the conditioning contexts so that the model structure enables exact inference with dynamic programming. Results show that the parser is more accurate than the feed-forward generative model of Chapter 3, while language modelling performance is competitive with vanilla RNNs. Furthermore we show that the model can learn parse trees without supervision. The research in chapter has been accepted for publication as Buys and Blunsom (2018).

Chapter 5: Neural Semantic Graph Parsing

In this chapter we propose a neural encoder-decoder RNN model to parse semantic graphs. In particular we propose a transition-based model within the encoder-decoder framework. The model includes an attention mechanism, an architecture that models the alignment between sentence tokens and graph nodes explicitly, and a stack-based architecture for the transition system. We report experimental results for parsing two semantic representations: Abstract Meaning Representation (AMR) and graphs based on Minimal Recursion Semantics. Our parser obtains a graph overlap F1 score of 84.2% on parsing Minimal Recursion Semantics, which is an improvement of 4.5% over an attention-based baseline. While our model is less accurate than the grammar-based parser on a test set consisting only of sentences which the grammar is able to parse, it is an order of magnitude faster. On AMR parsing we obtain 60.1% F1, which is higher than previous neural AMR parsers and other parsers using comparable resources. This chapter is based on the publications Buys and Blunsom (2017a) and Buys and Blunsom (2017b).

Chapter 6: Conclusion

We summarize our results, reflect on the extent to which the aims of this thesis have been accomplished and discuss the conclusions we can draw from that. We conclude by discussing directions for future work.

Chapter 2

Background: Language modelling and parsing

In this chapter we introduce the probabilistic models and algorithms that form the basis of the models developed in this thesis. *Language modelling* refers to a class of models that estimate probability distributions over words in sentences. We present three models: A count-based approach which employs Bayesian smoothing to interpolate between different conditioning contexts to estimate next word probabilities; a feed-forward neural network that uses a fixed-size context; and a recurrent neural network with unbounded context. These models will form the basis of the generative syntactic models we develop in Chapters 3 and 4. Next we introduce syntactic representations, in particular dependency graphs. *Transition-based parsing* is a class of stack-based algorithms that parse sentences incrementally to trees or graphs. We present two transition systems for projective syntactic graphs: arc-standard and arc-eager. We also present an extension to arc-eager for parsing semantic graphs. Finally we review models for supervised and unsupervised dependency parsing.

2.1 Language modelling

Let w be a word, a discrete symbol such that $w \in V$, where the vocabulary V is an ordered set. Words are represented interchangeably by their integer indexes in V. A *sentence* $\mathbf{w}_{1:n}$ is a sequence of words of length n, where w_n is equal to a designated end-of-sentence symbol, denoted as `EOS`. We shall let w_0 always be equal to a designated start-of-sentence symbol. The goal of language modelling is to estimate the probability distribution $p(\mathbf{w}_{1:n})$, which factorises as a product of multinomials:

$$p(\mathbf{w}_{1:n}) = \prod_{i=1}^{n} p(w_i|\mathbf{w}_{0:i-1}). \tag{2.1}$$

The first paradigm that we consider (Sections 2.1.1 and 2.1.2) is n-gram language modelling. An n-gram is a sequence of n words. The frequency counts of n-grams in a corpus are the sufficient statistics for estimating an n-gram model from that corpus, where $n \geq 2$. An n-gram model estimates the word probabilities in Equation 2.1 by making a Markov assumption: The distribution over the next word in the sentence depends on the $n-1$ preceding words, but is conditionally independent of the rest. The multinomial distribution over w_i is estimated as

$$p(w_i|\mathbf{w}_{1:i-1}) \approx p(w_i|\mathbf{w}_{\max(i-n+1,0):i-1}). \tag{2.2}$$

The conditioning context will also be denoted as $\mathbf{h}$, where $\mathbf{h} \in V^{n-1}$ if $i \geq n$.

2.1.1 Non-parametric Bayesian modelling

The first approach to estimating the distribution in Equation 2.2 is *count-based* methods. Language is inherently sparse: Empirically the relation between word types and the frequency of their occurrence in a document follow a power-law distribution. In particular Zipf's law (Zipf, 1949) states that the frequency of a word is inversely proportional to its rank in the frequency table. Therefore direct maximum likelihood estimation is unfeasible even for small values of n.

Smoothing refers to a number of techniques which address this problem (see Chen and Goodman (1999) for an overview). Smoothing methods adjust the counts of low-frequency n-grams, which are unreliable – this is referred to as discounting. They also employ *backoff*, using the estimates from smaller values of n when an n-gram is unknown, or *interpolate* different orders of n-gram estimates. The best performing and most widely used smoothing method is interpolated Kneser-Ney smoothing (Kneser and Ney, 1995; Chen and Goodman, 1999). Count-based models were the state of the art for language modelling for almost two decades, before being overtaken by recurrent neural networks. They are widely used in applications such as automatic speech recognition, phrase-based machine translation and text prediction for mobile keyboards.

We present a Bayesian language model (Goldwater et al., 2006b; Teh, 2006a) based on hierarchical Pitman-Yor Processes (HPYPs), of which an approximation recovers interpolated Kneser-Ney smoothing. The advantage of the Bayesian formulation is that inference (based on Gibbs sampling) extends straightforwardly to semi-supervised or unsupervised learning; we shall see an application in Chapter 3. HPYPs have been applied to various unsupervised NLP tasks, including word segmentation (Goldwater et al., 2006a), Parts of Speech induction (Blunsom and Cohn, 2011) and grammar induction (Blunsom and Cohn, 2010).

The Pitman-Yor process The definitions that follow are based on Teh (2006b) and Teh (2006a). The Pitman-Yor process (PYP) is a generalisation of the Dirichlet process, which defines a process generating distributions over a probability space X, with discount parameter $0 \leq d < 1$, strength parameter $\theta > -d$ and base distribution B. A draw of a posterior distribution G from a Pitman-Yor process is denoted as $G \sim \text{PYP}(d, \theta, B)$.

The Chinese restaurant process (CRP) characterises a sequence of draws $x_1, \ldots, x_n$ from G with the analogy of a Chinese restaurant: The restaurant has an infinite number of tables; each has capacity to seat an infinite number of customers. Customers enter the restaurant one by one, each representing a draw from G. Let c_k be the number of customers sitting at table k and t the number of occupied tables. When the ith customer enters, he chooses at which table to sit according to the distribution

$$p(z_i = k|\mathbf{z}_{1:i-1}) = \begin{cases} \frac{c_k - d}{i-1+\theta} & \text{if } 1 \leq k \leq t, \\ \frac{td+\theta}{i-1+\theta} & \text{if } k = t+1, \end{cases} \tag{2.3}$$

where k is the index of the table chosen and $\mathbf{z}_{1:i-1}$ the current seating arrangement. Each table serves one dish, which is assigned with a draw from the base distribution B when it receives its first customer. It can be shown that the sequence generated by the Chinese restaurant process is exchangeable, i.e. the distribution is invariant to the ordering of the sequence (Pitman, 1995).

In language modelling, we can consider the PYP as a prior for unigram distributions: Let every dish correspond to a word type, and every customer to a token drawn from the distribution being constructed. The base distribution B is uniform distribution over the words in vocabulary. Customers sitting at the same table correspond to tokens of the same word type. Suppose that the training corpus has c_w occurrences of word w. Given a seating arrangement $\mathbf{z}_{1:i}$, the next word probability distribution is

$$p(x_{i+1} = w|\mathbf{z}_{1:i}) = \frac{c_w - d}{\theta + i} + \frac{\theta + td}{\theta + i}B(w). \tag{2.4}$$

When setting $d = 0$ the distribution reduces to the Dirichlet distribution, with θ acting as pseudo-counts. The model has a self-reinforcing property: The more often a word is drawn, the more likely it is to be drawn again, reflecting the power-law distribution of words.

Hierarchical Pitman-Yor processes To construct an n-gram model based on a PYP prior, the model is extended to interpolate between different-sized contexts. The hierarchical Pitman-Yor process (HPYP) employs the PYP recursively as a prior over models of decreasing context size. For conditioning context $\mathbf{h}_{1:m-1}$ the HPYP is defined as

$$\begin{aligned} G_{\mathbf{h}_{1:m-1}} &\sim \text{PYP}(d_{m-1}, \theta_{m-1}, G_{\mathbf{h}_{1:m-2}}) \\ G_{\mathbf{h}_{1:m-2}} &\sim \text{PYP}(d_{m-2}, \theta_{m-2}, G_{\mathbf{h}_{1:m-3}}) \\ \ldots & \\ G_{\emptyset} &\sim \text{PYP}(d_0, \theta_0, \text{Uniform}), \end{aligned} \tag{2.5}$$

where d_k and θ_k are the discount and strength discount parameters for PYPs with conditioning context length k. Each back-off level drops one context element. The distribution for an empty context backs off to the uniform distribution over the vocabulary.

The Chinese restaurant analogy extends to the hierarchical PYP. Each restaurant corresponding to $G_{\mathbf{h}}$ is connected to a parent restaurant, corresponding to its base distribution $G_{\pi(\mathbf{h})}$, where $\pi(\mathbf{h}_{1:m}) = \mathbf{h}_{1:m-1}$. At each table the dish is drawn from the parent restaurant distribution by sending a customer to it; this is done recursively. In the hierarchy of restaurants every customer corresponds to a table at the level above (except for the top level) and every dish can be traced back to the base distribution B. In language modelling this means that every restaurant corresponds to a m-gram context and its parent restaurant is the $m-1$-gram context it backs off to.

The prior incorporates the belief that symbols in the context are ordered from most informative to least informative to the distribution. In an n-gram language model, the natural assumption is that the further away a context word is from the predicted word, the less informative it is. The less certainty the model has about its estimates, the more it will fall back on the uniform distribution.

Inference Given n-gram training data, the seating arrangement is not deterministic, as multiple tables can serve the same dish. Inference over seating arrangements is performed with Gibbs sampling, based on routines to add or to remove a customer from a restaurant, given the dish that the customer eats. An initial seating arrangement is sampled simply by sequentially adding each training example to the restaurant corresponding to its context, following the distribution given by the current seating arrangement. For subsequent iterations, Gibbs sampling is performed on training examples by iteratively removing a customer from its restaurant, and adding it back by sampling a table from the new distribution. The discount and strength parameters are sampled with slice sampling (Neal, 2003).

Blunsom et al. (2009) proposed efficient data structures to make inference practical without resorting to approximating the seating arrangement. Instead of representing each table explicitly, a histogram is maintained for each dish w_i in a restaurant. The histogram tracks how many tables with m customers are serving the dish, for all $m > 0$. The property of exchangeability makes it unnecessary to know at exactly which table each customer is sitting. When a customer is added, it is first decided whether they should join an existing table or start a new one. If the customer joins an existing table, the table number is sampled from the histogram and the histogram is updated. To remove a customer, the table from which to remove him is again sampled from the histogram distribution.

Interpolated Kneser-Ney smoothing can be recovered as a deterministic approximate inference procedure for the Bayesian model. All strength parameters $\theta_{|\mathbf{h}|}$ are set to 0, and the seating arrangement is made deterministic by assuming that each dish in each restaurant is served by only one table.

2.1.2 Feed-forward neural networks

An alternative approach to parameterising n-gram models to overcome sparsity is to use feed-forward neural networks (Bengio et al., 2003; Mnih and Hinton, 2007), based on the distributed representation of words as real-valued vectors.

Suppose we want to estimate $p(w|\mathbf{h})$, where $\mathbf{h} \in V^{m-1}$ as before. Each word h_j in the context is embedded as a vector $\boldsymbol{x}_j \in \mathbb{R}^d$ through a column lookup in embedding matrix $\boldsymbol{E} \in \mathbb{R}^{d \times |V|}$. Additional features over words can be incorporated through additive representations (Botha and Blunsom, 2014), using vector representations of the same dimension as the word embeddings. Example features include POS tags and morphological features. Suppose context feature f has an input representation $\boldsymbol{x}^{(f)} \in \mathbb{R}^D$. The composite representation is computed as $\boldsymbol{x} = \sum_{f \in \mu(w)} \boldsymbol{x}^{(f)}$, where $\mu(w)$ are the word features.

In a language model and other generative models all features have to be generated by the model (assigned probability) before they can be conditioned on. For example, to use POS tags as features, the model has to estimate the joint distribution over words and POS tags.

The projection layer is defined as

$$\phi(\mathbf{h}) = \sigma(\sum_{j=1}^{m-1} \boldsymbol{W}^{(j)} \boldsymbol{x}_j), \tag{2.6}$$

where $\boldsymbol{W}^{(j)} \in \mathbb{R}^{D \times D}$ are transformation matrices defined for each position in the n-gram context, and σ is an element-wise function. If the transformation matrices are square they can be approximated to be diagonal to reduce the number of model parameters. If σ is the identity function, the network is a log bilinear model. For neural networks, σ is a non-linearity such as the sigmoid logistic function, $\tanh$ or a rectifier.

This projection layer is then mapped to an output vector (called a logit)

$$\boldsymbol{v} = \boldsymbol{R}\phi(\mathbf{h}) + \boldsymbol{b}, \tag{2.7}$$

where $\boldsymbol{R} \in \mathbb{R}^{|V| \times D}$ is a matrix containing the output embeddings of each word and b is a bias vector.

The probability distribution is then estimated as

$$p(w = i|\mathbf{h}) = \frac{\exp(v_i)}{\sum_{j=1}^{|V|} \exp(v_j)}. \tag{2.8}$$

The function applied to $\boldsymbol{v}$ in Equation 2.8 is known as the *softmax* function and can be denoted as $\mathrm{softmax}_i(\boldsymbol{v})$.

Neural networks can be contrasted with traditional log-linear models, also known as maximum entropy models (Berger et al., 1996), where ϕ maps $\mathbf{h}$ to a high-dimensional vector space, typically with indicator functions, leading to a sparse vector. In log-linear models features are hand-designed, and include combinations of elementary features, while a neural network can learn arbitrary feature combinations. Log-linear models have been used in a wide range of NLP tasks, including part-of-speech tagging (Ratnaparkhi, 1996), language modelling (Rosenfeld, 1996) and named entity recognition (Borthwick, 1999). However, recently neural networks have obtained superior performance on most of these tasks.

Vocabulary factorisation One challenge with neural language models is the computational cost of computing the softmax over the entire vocabulary, during training and testing. An effective way to reduce this problem is class-based factorisation (Goodman, 2001; Baltescu and Blunsom, 2015). We use Brown clustering (Brown et al., 1992), an agglomerative clustering method that has proved to be very successful at clustering syntactically and semantically similar words, to cluster the vocabulary into approximately $\sqrt{|V|}$ classes. This formulation reduces the number of items in the softmax denominator, reducing the output layer time complexity from $O(|V|)$ to $O(\sqrt{|V|})$.

Let C be the set of classes and $\Gamma(c)$ be the set of words in class $c \in C$. The word probability factors as

$$P(w|\phi(\mathbf{h})) = P(c|\phi(\mathbf{h}))P(w|c, \phi(\mathbf{h})), \tag{2.9}$$

where c is the (unique) class of word w.

The class prediction logit is defined as

$$\boldsymbol{u} = \boldsymbol{S}\phi(\mathbf{h}) + \boldsymbol{d}, \tag{2.10}$$

where $\boldsymbol{S} \in \mathbb{R}^{D \times |C|}$ is the class embedding matrix and $\boldsymbol{d}$ a bias vector.

The word prediction logit for $\Gamma(c)$ is

$$\boldsymbol{v}_c = \boldsymbol{R}^{(c)}\phi(\mathbf{h}) + \boldsymbol{b}^{(c)}, \tag{2.11}$$

where $\boldsymbol{R}^{(c)}$ is the embedding matrix of words in class $\Gamma(c)$.

The probabilities are then

$$p(c|\phi(\mathbf{h})) = \mathrm{softmax}_c(\boldsymbol{u}), \tag{2.12}$$

$$p(w|c, \phi(\mathbf{h})) = \mathrm{softmax}_w(\boldsymbol{v}_c). \tag{2.13}$$

Training Neural language models are trained by minimising the negative log likelihood,

$$-\log p(\mathbf{w}_{1:n}) = -\sum_{i=1}^{n} \log p(w_i|\mathbf{h}_i), \tag{2.14}$$

over all the sentences in the training corpus. To minimise this objective a gradient-based optimisation algorithm is used. Due to the size of the parameter vectors second-order optimisation methods cannot be used, so we use (first-order) gradient descent. The gradients are computed at each layer as a function of the gradients and values at other layers. This approach to recursively computing gradients and updating parameters with gradients descent is referred to as *backpropagation* (Rumelhart et al., 1986). To train feed-forward networks we use mini-batch stochastic gradient descent (SGD) (Bottou, 1998) with AdaGrad (Duchi et al., 2011) and L2 regularisation.

2.1.3 Recurrent neural networks

While feed-forward neural networks are more powerful than count-based approaches, they still only make use of a fixed-sized context at each time step, restricting their ability to capture long-range dependencies.

Recurrent neural networks (RNNs) (Jordan, 1986; Elman, 1990) overcome this limitation. RNNs have been shown to obtain better intrinsic language modelling performance than feed-forward models (Mikolov et al., 2010; Jozefowicz et al., 2015, 2016).

In an RNN the hidden state at each time step depends on the previous hidden state, so the hidden state vector $\boldsymbol{h}_i$ represents the entire context from positions 1 to i. Suppose that the word embedding of w_i is $\boldsymbol{x}_i$ as before. The hidden state is computed as

$$\boldsymbol{h}_i = \mathrm{RNN}(\boldsymbol{x}_i, \boldsymbol{h}_{i-1}), \tag{2.15}$$

where RNN is a function mapping the input vector and previous hidden state to the new hidden state. Together the hidden state and recurrent function are also referred to as a *cell*.

There are multiple definitions of the recurrent function. A simple RNN cell (Elman, 1990) is defined as

$$\boldsymbol{h}_i = \sigma(\boldsymbol{W}^{(hx)}\boldsymbol{x}_i + \boldsymbol{W}^{(hh)}\boldsymbol{h}_{i-1} + \boldsymbol{b}^{(h)}). \tag{2.16}$$

Simple RNNs can suffer from vanishing or exploding gradient problems when trained with backpropagation (Bengio et al., 1994). Gradients are propagated from every time step back to the start of the sequence, and by expanding the chain rule the gradient becomes a product of the gradient with respect to each time step, which can become vanishingly small (or very large). A related problem is that RNN's ability to model long-distance dependencies in practice is limited. Solutions include using gates to control how much information is propagated between steps, and augmenting the RNN cell with a memory vector that can capture long-distance features.

The most widely-used RNN architecture following these suggestions is called Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997). LSTMs use a number of gates, including an input, output and forget gate, which controls how much weight to assign to the current input and how much to the representation of previous inputs. The standard LSTM cell is defined as

$$\boldsymbol{i}_i = \sigma(\boldsymbol{W}^{(ih)}\boldsymbol{h}_{i-1} + \boldsymbol{W}^{(ix)}\boldsymbol{x}_i + \boldsymbol{b}^i) \tag{2.17}$$

$$\boldsymbol{f}_i = \sigma(\boldsymbol{W}^{(fh)}\boldsymbol{h}_{i-1} + \boldsymbol{W}^{(fx)}\boldsymbol{x}_i + \boldsymbol{b}^f) \tag{2.18}$$

$$\boldsymbol{o}_i = \sigma(\boldsymbol{W}^{(oh)}\boldsymbol{h}_{i-1} + \boldsymbol{W}^{(ox)}\boldsymbol{x}_i + \boldsymbol{b}^o) \tag{2.19}$$

$$\boldsymbol{z}_i = \tanh(\boldsymbol{W}^{(zh)}\boldsymbol{h}_{i-1} + \boldsymbol{W}^{(zx)}\boldsymbol{x}_i + \boldsymbol{b}^z) \tag{2.20}$$

$$\boldsymbol{c}_i = \boldsymbol{f}_i \circ \boldsymbol{c}_{i-1} + \boldsymbol{i}_i \circ \boldsymbol{z}_i \tag{2.21}$$

$$\boldsymbol{h}_i = \boldsymbol{o}_i \circ \tanh(\boldsymbol{c}_i), \tag{2.22}$$

where $\boldsymbol{i}_i$, $\boldsymbol{f}_i$ and $\boldsymbol{o}_i$ are the input, forget and output gates respectively, $\boldsymbol{c}_i$ is the memory cell and $\circ$ represents element-wise multiplication.

The RNN language model is constructed by adding an output layer over the hidden state at each time step, followed by a softmax:

$$\boldsymbol{v}_i = \boldsymbol{W}^{(vh)}\boldsymbol{h}_i + \boldsymbol{b}^v \tag{2.23}$$

$$p(w_{i+1} = w|\mathbf{w}_{0:i}) = \text{softmax}_w(\boldsymbol{v}_i) \tag{2.24}$$

The RNN is trained with SGD, similar to feed-forward networks.

2.2 Syntactic and semantic representations

In this thesis we focus on *dependency*-based representations of syntactic and semantic structure. Dependency graphs represent the structure of sentences through arcs between words, usually labelled with relations.

2.2.1 Dependency graphs

A *dependency graph* (Nivre, 2008) is a directed graph $G = (N, A)$, where $N = \{0, \ldots, n-1\}$ is a set of nodes, $A \subseteq N \times L \times N$ is a set of labelled directed arcs, and L a set of dependency labels. For syntactic dependency graphs there is a one-to-one relation between nodes 1 to $n-1$ and the words in sentence $\mathbf{w}_{1:n-1}$. A node corresponding to the end-of-sentence symbol w_n may be predicted by a parser, but does not form part of the graph.

Each arc, denoted as $(i, l, j) \in A$, encodes a dependency between two nodes, where i is the head, j the dependent and l is the dependency type of j. An arc can also be written as $i \xrightarrow{l} j$. Node 0 is the designated root node, i.e. there is no node i and a label l such that $(i, l, 0) \in A$. Syntactic parses are trees: the graph is acyclic, connected and every node has at most one head, i.e. if $(i, l, j) \in A$ then there is no node i' and label l' such that $(i', l', j) \in A$ and $i \neq i'$ or $l \neq l'$. Semantic graphs do not have to obey these constraints.

A dependency graph G is *projective* if and only if, for each arc $(i, l, j) \in A$ and node $k \in N$, where $i < k < j$ or $j < k < i$, there is a subset of arcs $\{(i, l_1, i_1), \ldots, (i_{m-1}, l_m, i_m)\}$ in A such that $i_m = k$. A projective dependency graph can be drawn on a plane such that there will be no crossing arcs, as in Figures 2.1 and 2.2.

2.2.2 Dependency representations

We next discuss the syntactic dependency representations we will use. Annotated dependency treebanks can be obtained either by direct manual annotation or by automatically converting constituency parses or other dependency parses through hand-crafted rules, optionally followed by manual verification. The treebanks for the English syntactic dependency representations we use in this thesis were obtained through rule-based conversion.

In many other languages, especially those with somewhat free word order (e.g. Czech) dependency syntax is a more natural representation than constituency structure, as the syntax is less dependent on the word order; consequently dependency treebanks have been developed for these languages directly.

The first representation that we introduce is based on the head-finding rules of Yamada and Matsumoto (2003). We refer to this representation as YM. The conversion is performed with the Penn2Malt converter.[1] YM has 12 dependency labels. An example YM

[1]http://stp.lingfil.uu.se/ nivre/research/Penn2Malt.html

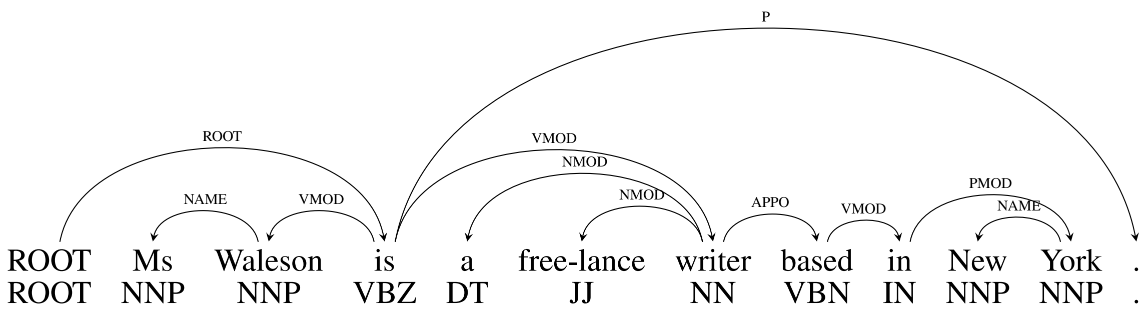

P
ROOT
VMOD
NMOD
NAME
VMOD
NMOD
APPO
VMOD
PMOD
NAME
ROOT Ms Waleson is a free-lance writer based in New York .
ROOT NNP NNP VBZ DT JJ NN VBN IN NNP NNP .

Figure 2.1: A YM dependency tree. The words and POS tags of the sentence are given. Arcs indicate *head* → *modifier* dependencies, and are labelled with dependency relations.

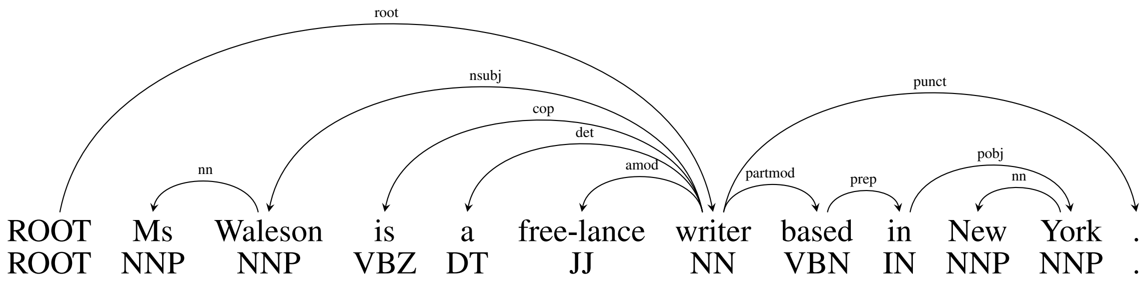

root
nsubj
punct
cop
det
pobj
amod
nn
partmod
prep
nn
ROOT Ms Waleson is a free-lance writer based in New York .
ROOT NNP NNP VBZ DT JJ NN VBN IN NNP NNP .

Figure 2.2: A Stanford dependency tree.

dependency tree is given in Figure 2.1.

The CoNLL syntactic dependency representation (CD) (Johansson and Nugues, 2007)[2] is a more sophisticated representation following the same principles as YM. However it has not been used widely for reporting parsing results, and contains some semantically-motivated non-projective dependencies, while we focus on projective syntactic dependencies.

The other dependency conversion that we shall use is the Stanford dependency representation (SD) (De Marneffe and Manning, 2008). This representation is obtained through version 3.3.0 of the Stanford parser.[3] The Stanford dependencies have a richer dependency label set than YM, with 45 labels; the converter uses more elaborate hand-crafted rules based on linguistic knowledge, compared to Penn2Malt. An SD dependency tree is given in Figure 2.2.

[2]Implemented in the LTH converter: http://nlp.cs.lth.se/software/treebank_converter/

[3]http://nlp.stanford.edu/software/lex-parser.shtml

There are a number of other differences between the representations: The Stanford representation prefers to let content words between which a relation exists modify each other directly, while YM is more motivated by traditional syntactic considerations. Copula verbs (including forms of *to be*) are treated as the head of the clause in YM (see Figure 2.1) while SD treats the complement (*writer* in the example) as the head of the sentence.[4] YM chooses the auxiliary verb to be the head of the main lexical verb in a clause, while SD lets the main verb be the head.

The first effort to collect standardized dependency treebanks was carried out in the context of the CoNLL 2006 and 2007 shared tasks (Buchholz and Marsi, 2006; Nivre et al., 2007). The more recent Universal Dependencies (UD) project has the goal of harmonizing dependency annotations across languages to enable cross-lingual comparison and transfer (Nivre et al., 2016). The most notable design decision of UD is that it goes even further than SD in giving preference to dependencies between content words: A preposition modifies the main noun in a prepositional phrase, rather than acting as its head. This enables harmonization with languages where the syntactic equivalent of function words in English is expressed through morphology rather than separate words.

There are also *semantic* dependency formalisms. A number of such representations for English, derived from more linguistically expressive representations, have been released with the SemEval shared task on semantic dependency parsing (Oepen et al., 2014, 2015). We shall introduce linguistically expressive semantic representations in Chapter 5.

2.2.3 Constituency syntax

In order to understand the linguistic and algorithmic motivations behind dependency parsing, we place it in the context of constituency parsing.

Chomsky (Chomsky, 1957) proposed the use of constituency (phrase structure) parse trees to describe the syntactic structure of English sentences. A number of syntactic theories based on constituency structure were subsequently developed, including transformational grammar (Chomsky, 1965), Government and Binding theory (Chomsky, 1981) and Head-driven Phrase Structure Grammar (Pollard and Sag, 1994).

[4]There is also a version of SD which selects copula verbs as heads.

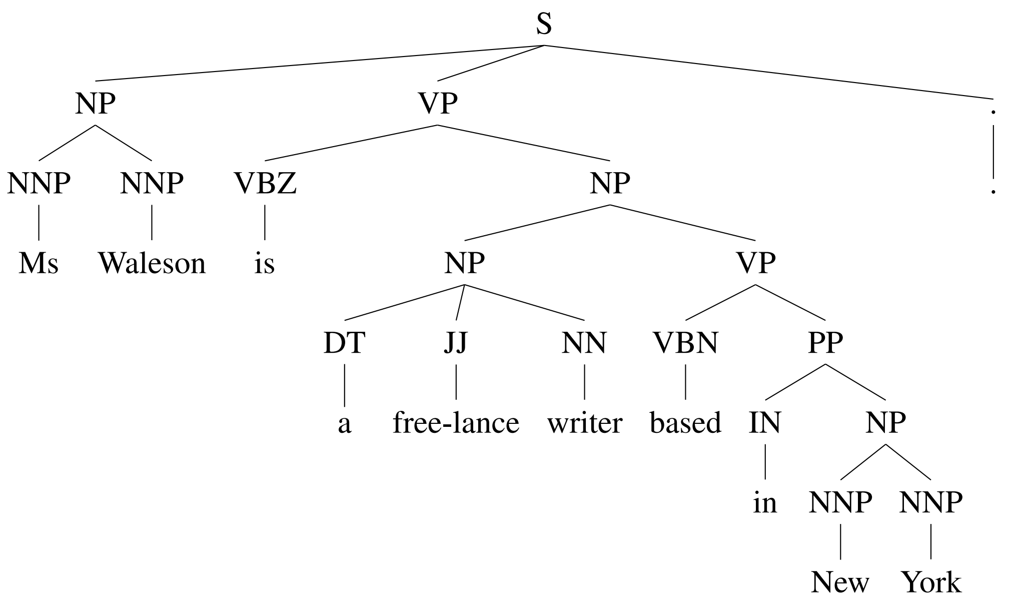


Figure 2.3: A constituency parse tree. The words of the sentence are at the leaves of the tree. The pre-terminal nodes are POS tags and the other nodes label constituency phrases in the sentence.

To facilitate syntactic modelling in computational linguistics treebanks, datasets of syntactically annotated sentences were developed. In English, the most commonly used one is the Wall Street Journal (WSJ) section of the Penn treebank (PTB) (Marcus et al., 1993), consisting of approximately 1 million words of newspaper text. An example constituency parse tree is given in Figure 2.3. The words of the sentence are the terminal (leaf) nodes of the tree, while non-terminal nodes each span a constituent phrase. The pre-terminals (parent nodes of the terminals) are parts-of-speech (POS) tags. The PTB also contains additional information which is usually omitted for parsing: Function tags are optional labels of non-terminals that represent grammatical functions of phrases, such as subject, predicate, or adverbial functions such as direction, extend and manner. Null elements are terminals which are not words but represent additional grammatical structure (according to the underlying linguistic theory), such as the movement of words out of their canonical positions.

Constituency trees can be generated by context-free grammars (CFGs) (Chomsky, 1959). Context-free grammars can be parsed by cubic time algorithms, of which the most widely used is the CKY algorithm, a bottom-up parsing algorithm based on dynamic programming. Probabilistic CFGs are top-down generative models of constituency trees. However, simply reading a context-free grammar off parse trees and taking maximum likelihood estimates of rule weights does not lead to accurate parsing, due to the strong independence assumptions that this model makes.

Grammar-based constituency parsers are based on either lexicalisation (Collins, 1997; Bikel, 2004; Charniak and Johnson, 2005) – annotating phrase labels with their head words, or refinement of non-terminal labels without lexicalisation (Klein and Manning, 2003; Petrov et al., 2006). Both of these approaches have to deal with computational complexity, controlling the size of the grammar, and sparsity in estimating rule probabilities.

Discriminative constituency parsers score candidate phrases conditioned on the words in the sentence being observed. Models can be trained either with a max-margin objective (Taskar et al., 2004), or as conditional random fields (globally-normalised log linear models) (Petrov and Klein, 2007). However, training discriminative models is more expensive, as they have to perform inference over trees during training.

There are a number of motivations for focussing on dependency rather than constituency parsing.

- Dependencies are linguistically plausible in all languages, while constituency parsing, by making implicit assumptions about the relation between word order and syntactic structure, might not be.

- Developing dependency annotations in a new language is simpler than for constituency structure, which is typically linked to the development of a grammar or a specific linguistic framework for that language.

- Dependency parsers are usually faster than constituency parsers. This is especially due to the absence of a large grammar constant in dependency parsing, which dominates constituency parsing cost.

Note that we do not claim that dependency parsing has full linguistic expressivity – there are clearly some intricate constructions that can be expressed better with more linguistically expressive grammars. In the realm of semantics we'll turn to more expressive representations. However, for syntactic parsing dependencies offer a better trade-off between expressivity, usability and scalability than constituency for most applications and languages.

2.3 Transition systems for dependency parsing

Next we present standard algorithms for transition-based dependency parsing (Nivre, 2008). These algorithms are related to shift-reduce parsing for context-free grammars, originally developed in the context of compiler theory (Knuth, 1965; Aho et al., 1986).

A *transition system* is a tuple $S = (C, T, c^i, C_t)$, where C is a set of *configurations*, T is a finite set of transitions, each of which is a partial function $t : C \rightarrow C$, c^i is the initial configuration, and $C_t \subseteq C$ is a set of terminal configurations (Nivre, 2008).

A configuration is a tuple $c = (\sigma, \beta, A)$, where the stack σ is a list of nodes from the set of nodes N, the buffer β a single node not in σ, and A a set of arcs. Nodes are represented by their indexes. The top stack element is referred to as σ_1, and subsequent elements as $\sigma_2, \sigma_3, \ldots, \sigma_{|\sigma|}$. By convention the stack is written with its topmost element to the right. Concatenation is indicated by a vertical bar: $\sigma|j$ indicates that j is the top of the stack.

The initial configuration c^i is $([0], 1, \emptyset)$. The set of terminal configurations C_t consists of all configurations of the form $([0], n, A)$, where A is a set of arcs and w_n is the end-of-sentence symbol. Given a sentence w, a parser based on transition system S processes w from left to right. At each step it applies a transition, until it reaches a terminal configuration. A sequence of configurations $\gamma = c_0, \ldots, c_m$ is *complete* when $c_0 = c^i$ and $c_m \in C_t$. The dependency graph defined by A in the terminal configuration is returned as the parse of w. Note that in general a parse may have multiple derivations, i.e., multiple transition sequences may result in the same parse. This is referred to as *spurious ambiguity*. A transition system can also be used as a generative model, where the transition actions also generate the set of nodes.

An *oracle* for a transition system S is a function $o(c, G)$ that maps the current configuration c and the gold parse G to the next transition that should be performed. Given an oracle for a sentence w we can deterministically parse the sentence by applying the oracle transition to the configuration at each time step, starting at the initial configuration, until a final configuration is reached. The oracle is used to train the model by finding transition sequences that generate its gold parse.

2.3.1 Arc-standard

The arc-standard transition system (Nivre and Scholz, 2004) has 3 transition types: shift (sh), left-arc (la$_l$) and right-arc (ra$_l$), where l is a dependency label. For labeled parsing there are $2|L| + 1$ transitions. The transitions are defined as follows:

$$\text{(sh)}\ \ (\sigma, i, A) \vdash (\sigma|i, i+1, A)$$

$$(\text{la}_l)\ \ (\sigma|i|j, \beta, A) \vdash (\sigma|j, \beta, A \cup \{(j, l, i)\})$$

$$(\text{ra}_l)\ \ (\sigma|i|j, \beta, A) \vdash (\sigma|i, \beta, A \cup \{(i, l, j)\})$$

The left-arc transition has a precondition $i \neq 0$ to ensure that node 0 is at the root of the tree. The left-arc and right-arc transitions remove the dependent of the added arc from the stack. Therefore arc-standard is a bottom-up parser, i.e. once a node is attached to its head, no further dependents can be added to the node.

Algorithm 1 Arc-standard oracle.

```
 1: Given $(\sigma, \beta, A), G$
 2: if $|\sigma| < 2$ then
 3:     return sh
 4: else
 5:     $j \leftarrow \sigma_1$
 6:     $i \leftarrow \sigma_2$
 7:     if $(j, l, i) \in G \wedge \neg\exists k \exists l'[(i, l', k) \in G \wedge \neg(i, l', k) \in A]$ then
 8:         return la$_l$
 9:     else if $(i, l, j) \in G \wedge \neg\exists k \exists l'[(j, l', k) \in G \wedge \neg(j, l', k) \in A]$ then
10:         return ra$_l$
11:     else if $\beta < n$ then
12:         return sh
13:     else
14:         return No valid action.
15:     end if
16: end if
```

We define an oracle for arc-standard parsing, which is a function of the current configuration and the gold tree G (see Algorithm 1). The oracle enforces the bottom-up property so that a (valid) arc is not added before the dependent has been attached to all its children. The transition system's spurious ambiguity can be characterised as follows: If there exists a valid arc between σ_2 and σ_1 and the oracle decides to shift, the same pair will only occur on top of the stack again after a right dependent has been attached to σ_1. Therefore the oracle has to add right arcs greedily if they are valid, while a valid left arc may be delayed if σ_1 has unattached right dependents in G. Our oracle follows standard practice and adds arcs as soon as they are valid.

We shall also experiment with a non-deterministic oracle which gives a set of valid actions instead of just one. In contrast to dynamic oracles (Goldberg and Nivre, 2013) that perform training with exploration, we are only interested in characterising the derivations of the correct parse tree, so the non-deterministic oracle can assume that for the given configuration there exists a derivation of the gold tree G.

As an example of arc-standard parsing we give the oracle derivation of the parse in Figure 2.1 in Figures 2.4 and 2.5. The example shows the transition system as a generative model, where *shift* generates the node shifted onto the stack.

2.3.2 Arc-eager

The arc-eager transition system (Nivre, 2003) has 4 transition types: shift (sh), left-arc (la$_l$), right-arc (ra$_l$), and reduce (re). They are defined as follows:

(sh) $(\sigma, i, A) \vdash (\sigma|i, i+1, A)$

(la$_l$) $(\sigma|i, j, A) \vdash (\sigma, j, A \cup \{(j, l, i)\})$

(ra$_l$) $(\sigma|i, j, A) \vdash (\sigma|i|j, j+1, A \cup \{(i, l, j)\})$

(re) $(\sigma|i, \beta, A) \vdash (\sigma, \beta, A)$

The left-arc and reduce actions are mutually exclusive. The precondition for left-arc is that the stack top does not already have a head. For reduce the precondition is that it already has a head. Both are invalid if the stack top is the root.

Left arcs are added bottom-up and right arcs top-down. The strategy is called *eager* because any pair of words will appear together at the top of the stack and on the buffer only once during a valid derivation. Therefore if an arc exists between the two, it must be added eagerly (in contrast to arc-standard, where the decision may have to be postponed in order to parse bottom-up). However, a parse may still have multiple derivations, due to

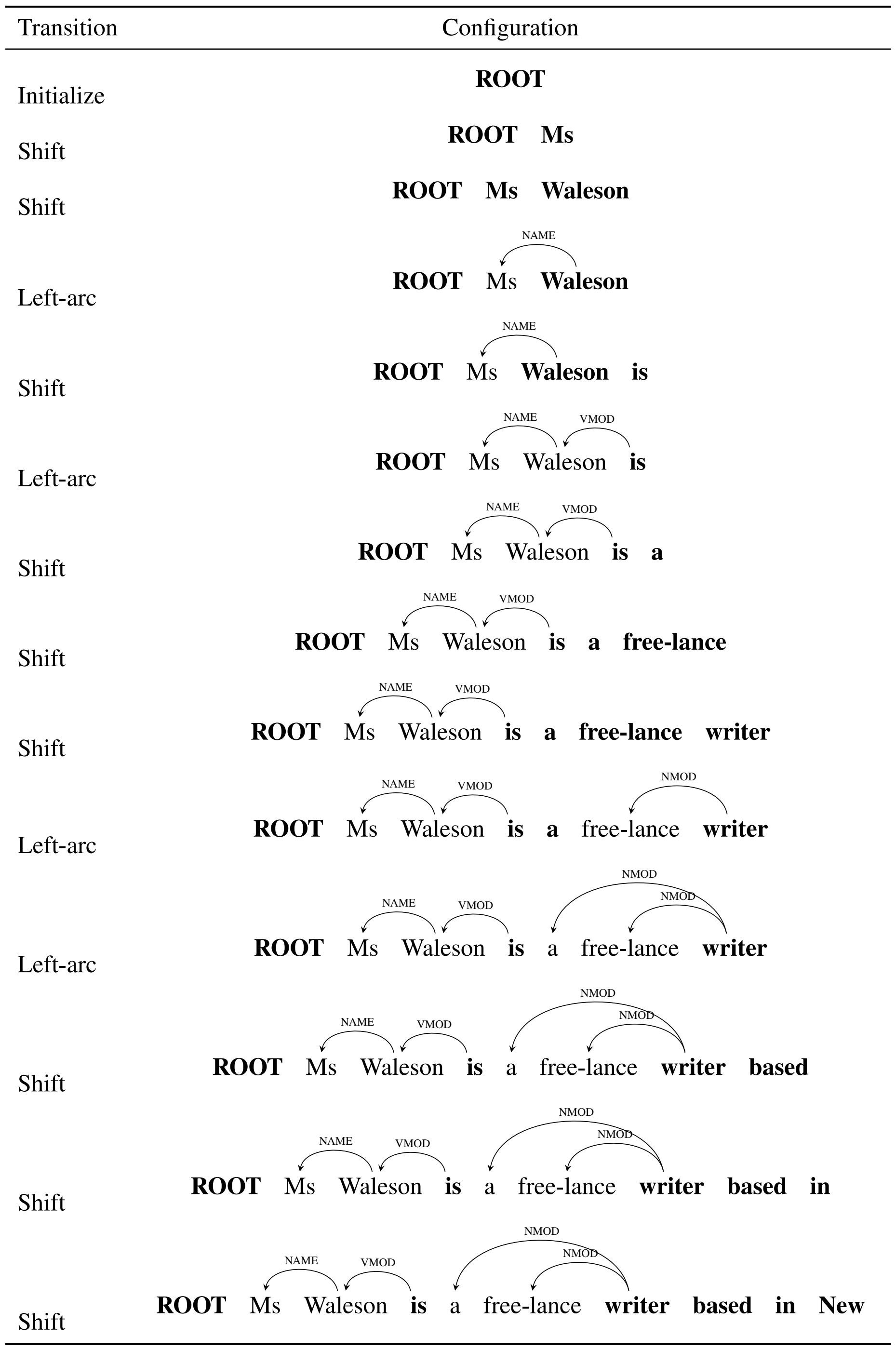


Figure 2.4: Arc-standard parsing derivation. At each step the transition taken and the words and arcs generated up to that step are shown. The stack consists of words in bold. The bottom-to-top order of nodes of the stack corresponds to the left-to-right word order.

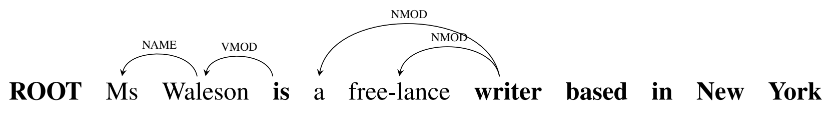
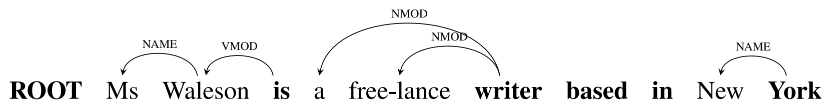
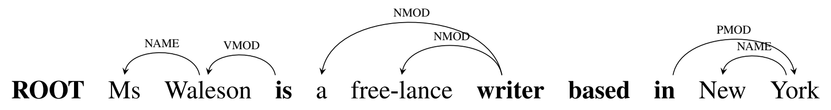
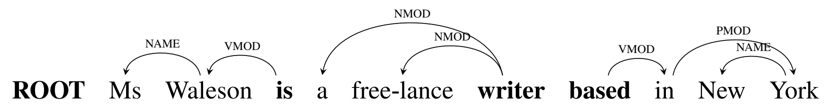
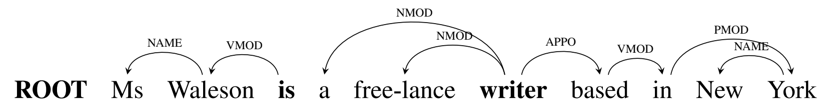
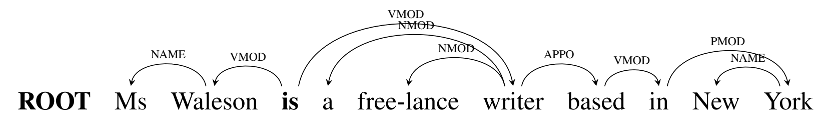
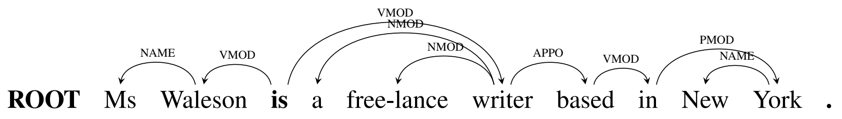
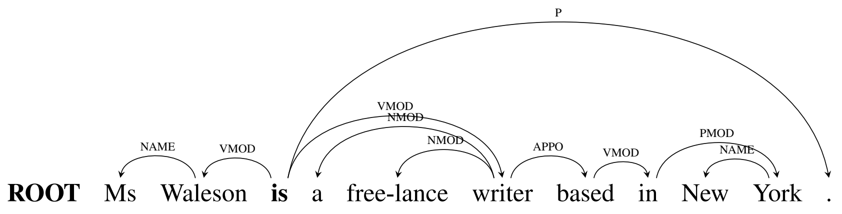
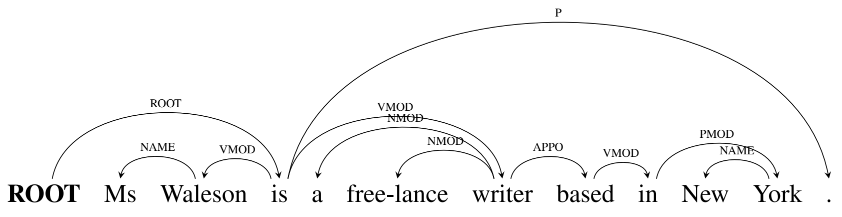
Transition	Configuration
Shift	
Left-arc	
Right-arc	
Right-arc	
Right-arc	
Right-arc	
Shift	
Right-arc	
Right-arc	

Figure 2.5: Arc-standard parsing derivation (continued).

ambiguity in performing shift or reduce in some cases where the top stack node already has a head and has been attached to all its dependents in the gold parse.

Again there is spurious ambiguity in the arc-eager model. In this case, the ambiguity lies in whether nodes on the stack should be reduced as soon as they have received all their dependents (early reduce), or if they should be kept until a reduce is necessary in order to add an arc between β and a node further down the stack (late reduce). Our oracle (Algorithm 2) performs early reduce. Discriminative oracles for arc-eager parsing typically use late reduce (Goldberg and Nivre, 2012).

As an example of arc-eager parsing we give the oracle derivation of the parse in Figure 2.1 in Figures 2.6 and 2.7. The example again shows the transition system as a generative model, where *shift* generates the node shifted onto the stack.

Algorithm 2 Arc-eager oracle.

```
1: Given $(\sigma, \beta, A), G$
2: $i \leftarrow \sigma_1$
3: $j \leftarrow \beta$
4: if $(j, l, i) \in G$ then
5:     return la$_l$
6: else if $(i, l, j) \in G$ then
7:     return ra$_l$
8: else if $\exists k \exists l[(k, l, i) \in A] \wedge \neg\exists k \exists l'[(i, l', k) \in G \wedge \neg(i, l', k) \in A]$ then
9:     return re
10: else if $j < n$ then
11:     return sh
12: else
13:     return No valid action.
14: end if
```

Transition system for dependency graph parsing A variant of the arc-eager transition system has been proposed to predict graphs (Sagae and Tsujii, 2008; Titov et al., 2009; Gómez-Rodríguez and Nivre, 2010). The notion of projectivity for trees is extended to *planar* graphs: Planar graphs also have no crossing edges when drawn above the nodes on a plane, but no restrictions are placed on the number of heads of any node.

The transition system for planar graphs uses the same set of transitions as projective graphs. However, left-arc and right-arc are redefined so that they do not perform shift or

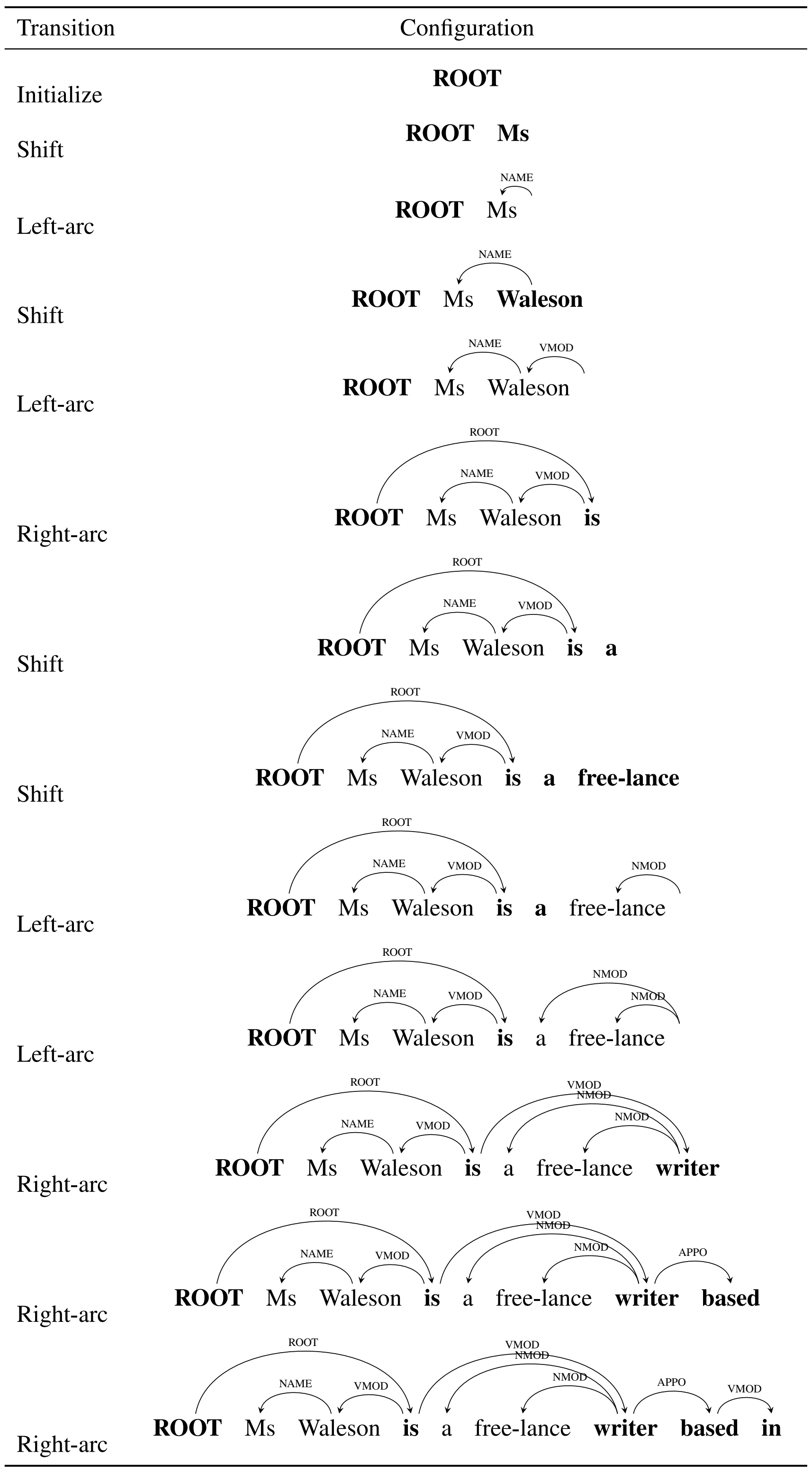


Figure 2.6: Arc-eager parsing derivation. At each step the transition taken and the words and arcs generated up to that step are shown. The stack consists of words in bold.

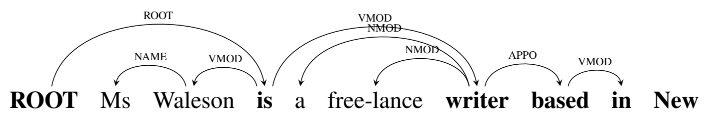
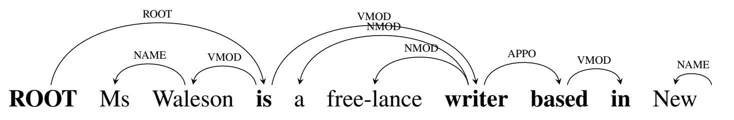
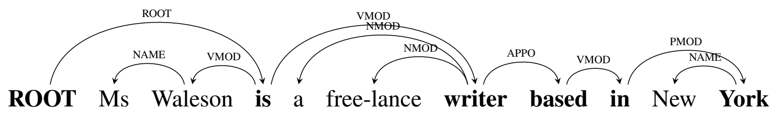
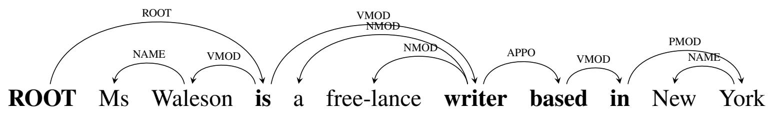
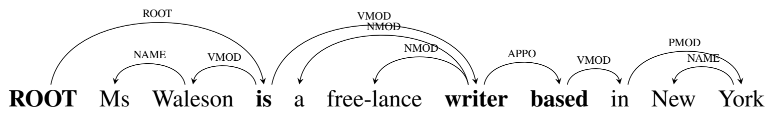
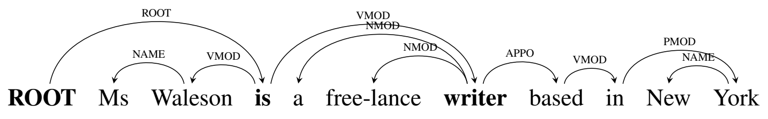
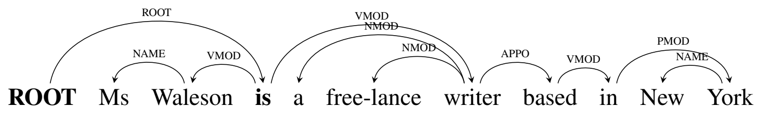
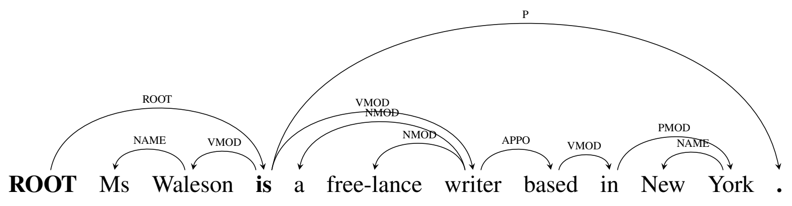
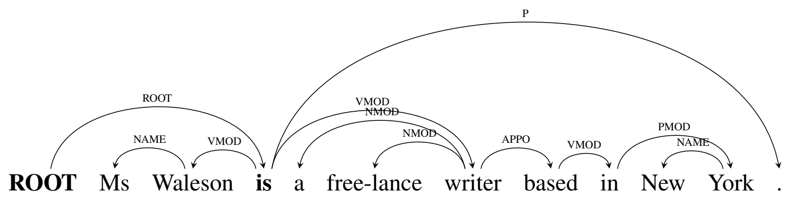
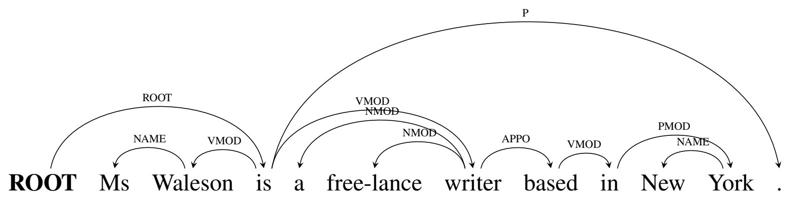
Transition	Configuration
Shift	 ROOT NAME VMOD VMOD NMOD NMOD APPO VMOD **ROOT** Ms Waleson **is** a free-lance **writer** **based** **in** **New**
Left-arc	 ROOT NAME VMOD VMOD NMOD NMOD APPO VMOD NAME **ROOT** Ms Waleson **is** a free-lance **writer** **based** **in** New
Right-arc	 ROOT NAME VMOD VMOD NMOD NMOD APPO VMOD PMOD NAME **ROOT** Ms Waleson **is** a free-lance **writer** **based** **in** New **York**
Reduce	 ROOT NAME VMOD VMOD NMOD NMOD APPO VMOD PMOD NAME **ROOT** Ms Waleson **is** a free-lance **writer** **based** **in** New York
Reduce	 ROOT NAME VMOD VMOD NMOD NMOD APPO VMOD PMOD NAME **ROOT** Ms Waleson **is** a free-lance **writer** **based** in New York
Reduce	 ROOT NAME VMOD VMOD NMOD NMOD APPO VMOD PMOD NAME **ROOT** Ms Waleson **is** a free-lance **writer** based in New York
Reduce	 ROOT NAME VMOD VMOD NMOD NMOD APPO VMOD PMOD NAME **ROOT** Ms Waleson **is** a free-lance writer based in New York
Right-arc	 P ROOT NAME VMOD VMOD NMOD NMOD APPO VMOD PMOD NAME **ROOT** Ms Waleson **is** a free-lance writer based in New York **.**
Reduce	 P ROOT NAME VMOD VMOD NMOD NMOD APPO VMOD PMOD NAME **ROOT** Ms Waleson **is** a free-lance writer based in New York .
Reduce	 P ROOT NAME VMOD VMOD NMOD NMOD APPO VMOD PMOD NAME **ROOT** Ms Waleson is a free-lance writer based in New York .

Figure 2.7: Arc-eager parsing derivation (continued).

reduce, but only add arcs. Furthermore, the preconditions are changed so that an arc can always be added as long as there isn't already an arc between the buffer and stack top. If the graph does not have to be connected, the pre-condition on reduce (that the stack top has to be headed) is dropped. The same oracle as for arc-eager parsing (Algorithm 2) can be applied to planar graphs (Gómez-Rodríguez and Nivre, 2010).

There have been a number of proposals for parsing particular classes of non-planar graphs. Titov et al. (2009) extend the transition system to add a *swap* action that swaps the two words on top of the stack. This has good, but incomplete coverage for non-planar semantic graphs. Gómez-Rodríguez and Nivre (2010) add a second stack to the transition system to parse 2-planar graphs, the class of non-planar graphs which can be decomposed into two planar graphs.

2.4 Syntactic dependency models

Supervised dependency parsers fall into two classes: Graph-based (scoring graph fragments) and transition-based (scoring transition actions). We also review unsupervised parsing and syntactic language modelling.

2.4.1 Graph-based parsing

Eisner (1996) proposed a generative graph-based dependency parser with a $O(n^3)$ dynamic programming-based parsing algorithm. The model generates dependency trees top-down, estimating the probability of dependents given their head and (where available) the previously generated adjacent sibling in the same direction. Backoff and smoothing are performed to reduce sparsity in the conditioning context of the distributions. Wallach et al. (2008) proposed a Bayesian HPYP parameterisation of this model, which increased performance, although it still underperforms comparable discriminative models.

The dynamic program in Eisner's algorithm is based on an arc-factored model for projective dependency trees: The score of the tree is the sum of the scores assigned independently to each head-dependent pair (corresponding to an arc). McDonald et al. (2005b) proposed a discriminative arc-factored model trained with MIRA, an online large-margin method for structured classification. Features are based on the word and POS tags of the head and dependent, as well as adjacent words in the sentence. For non-projective parsing the maximum spanning-tree (MST) algorithm can find the highest-scoring parse in $O(n^2)$ (McDonald et al., 2005a).

Graph-based models perform better if larger tree fragments are scored (as opposed to arc-factored models). Models for second-order (Mcdonald and Pereira, 2006; Carreras, 2007) and third-order parsing (Koo and Collins, 2010; Martins et al., 2013) have been proposed. However, non-projective parsing is NP hard for higher-order models (McDonald and Satta, 2007). Approximate inference methods have been proposed based on Integer Linear Programming (Martins et al., 2009) and dual decomposition (Koo et al., 2010; Martins et al., 2013). These methods are usually slow, but optimised implementations provide speed and accuracy comparable to transition-based parsers (Martins et al., 2013).

Recently it has been shown that arc-factored graph-based models based on bidirectional RNNs which encode each word in its sequential context can perform competitively (Kiperwasser and Goldberg, 2016), and reach state-of-the-art performance with better optimisation and hyperparameters (Dozat and Manning, 2017).

An alternative to directly scoring dependency graphs is to perform constituency parsing and transform the result to dependency parses with the rule-based converter used to obtain the dependencies in the first place. For English this still gives the most accurate results (Choe and Charniak, 2016; Kuncoro et al., 2017). However this approach is only possible in a limited number of languages, and is slower than direct dependency parsing (Kong and Smith, 2014).

2.4.2 Transition-based parsing

Most transition-based dependency parsers are based on discriminative classifiers. Early models trained local classifiers using k-nearest neighbour classification (Nivre and Scholz, 2004), support vector machines (SVMs) (Yamada and Matsumoto, 2003; Hall et al., 2006) or maximum-entropy or perceptron models. Further research focussed on feature engineering, developing feature templates resulting in millions of sparse features (Zhang and Nivre, 2011). Features are usually combinations of the words and POS tags of nodes on the stack and the buffer, as well as their parents, children and grandchildren.

While early models were based on greedy decoding, beam-search improves performance when the model is trained with a structured perceptron algorithm which performs beam-search during training (Collins and Roark, 2004; Zhang and Clark, 2008). In this way the model can learn to score non-optimal hypotheses that enter the beam. These models are normalised globally, instead of locally. Beam-search approximates searching for the optimal tree, so the structured perceptron can be seen as approximating a perceptron update against the optimal tree according to the model by performing early updating: As soon as the gold parse falls off the beam, an update is performed. A number of variants on how to perform perceptron updates have been proposed (Huang et al., 2012).

Locally normalised models suffer from the label bias problem (Lafferty et al., 2001). This limits their ability to revise earlier decisions (Andor et al., 2016), which explains why beam-search over locally normalised models results in only limited improvements. Locally normalised models with finite look-ahead can make limited inferences based on future information which might affect the current decision, while globally normalised models can learn to have lower confidence in its current decision.

An alternative approach to learn better greedy models is through dynamic and non-deterministic oracles (Goldberg and Nivre, 2012, 2013). This approach involves training with exploration - simulating the parser by making incorrect decisions during training and then learning to find the best possible action sequence after mistakes have been made. For some transition systems the optimal set of transition actions after an arbitrary sequence of transitions has been executed can be characterised exactly by non-deterministic oracles. Training the model with a non-deterministic, instead of a static oracle, in combination with exploration, has been shown to improve performance.

Finally, incremental parsing with inference based on particle filtering has been proposed as a model for human online sentence processing (Levy et al., 2009).

Neural network parsers Henderson (2003a,b, 2004) proposed a neural network approach to incremental left-corner constituency parsing using simple synchrony networks, a form of recurrent neural network that has additional connections to previous recurrent states based on the configuration of the incremental parser at each time step. The generative version of this model is more accurate than the discriminative one.

Titov and Henderson (2007) introduced a generative latent variable model for arc-eager transition-based parsing. This model is based on incremental sigmoid belief networks, which can be seen as neural networks where the hidden units are latent binary variables. Exact inference is intractable, so recurrent neural networks and variational mean field methods are proposed to perform approximate inference. The network architecture is similar to that of Henderson's constituency parsers. Training and decoding is very slow, and the model is less accurate than state-of-the-art discriminative dependency parsers.[5]

Chen and Manning (2014) proposed a greedy, locally-normalised feed-forward neural network based on distributed word representations. Weiss et al. (2015) showed that with careful hyperparameter optimisation performance of this approach can be improved to reach state-of-the-art performance. Furthermore Weiss et al. (2015) and Andor et al. (2016) showed that globally normalised training with a neural structured perceptron further improves performance. Dyer et al. (2015) proposed a *stack* LSTM which encodes the entire transition system stack with an RNN – the subtrees rooted at each node on the stack are encoded recursively. This model obtains high accuracy with greedy decoding.

2.4.3 Unsupervised parsing

The goal of unsupervised parsing, also known as grammar induction, is to learn parse trees directly from sentences, without annotations. Most models for unsupervised parsing are generative, although methods based on search-based structured prediction have also been proposed (Daumé III, 2009).

The first model to learn more accurate trees than an uninformative right-branching baseline was the dependency model with valence (DMV) (Klein and Manning, 2004). In this model the tree structure is generated top-down, similar to Eisner (1996), but with a simpler conditioning context that includes the valency of the head node – whether any dependents have already been generated. Crucially, parameters are initialised with the harmonic initialiser, where the probability of a word-dependent pair is inversely proportional to the distance between the words. The model is trained with expectation maximisation (EM). A number of extensions to this model use better priors and improved initialisation: Structural

[5]While the model would be more efficient if re-implemented on recent hardware and software, its neural network architecture is still very complex.

annealing starts with a strong preference for short dependencies and gradually relaxes that preference over time (Smith and Eisner, 2006). Gradually increasing the length of sentences used for training has a similar effect (Spitkovsky et al., 2010a). Viterbi EM training improves performance for longer sentences, when a good distribution for short sentences has already been found (Spitkovsky et al., 2010b). Models with richer contexts (Headden et al., 2009), logistic normal distributions (Cohen et al., 2008) and Tree Substitution Grammars to learn larger tree fragments (Blunsom and Cohn, 2010) have also been proposed. The typical setup limits the length of training sentences, assumes that gold part-of-speech (POS) tags are available, and learns models based only on POS tags or weak lexicalisation (including only words with a frequency above a relatively high threshold). However it has been shown that word-based grammar induction on large corpora can overcome the initialisation sensitivity of models trained on POS tags only (Pate and Johnson, 2016). Due to the strong independence assumptions that these models make they are not suitable for either accurate supervised parsing or language modelling.

2.4.4 Syntactic language modelling

In principle any generative parser can be applied to language modelling by marginalising over all possible parses of a sentence. However PCFGs make too strong independence assumptions for language modelling. Lexicalised PCFGs (Roark, 2001; Charniak, 2001) do show improvements over n-gram models, but decoding is prohibitively expensive for practical integration in language generation applications.

Chelba and Jelinek (2000) and Emami and Jelinek (2005) proposed incremental syntactic language models. Their models predict binarised constituency trees with a transition-based model, and are parameterised by deleted interpolation n-gram smoothing and feed-forward neural networks, respectively. Rastrow et al. (2012) applied a transition-based dependency language model to speech recognition, using hierarchical interpolation and relative entropy pruning. However, the model perplexity only improves over an n-gram model when interpolated with one.

Chapter 3

Markov Models for Generative Dependency Parsing

In this chapter, we propose generative models for transition-based dependency parsing based on Markov assumptions, decomposing the probability distribution into local predictions conditioned on bounded-length finite contexts. We show that these models are expressive enough for both parsing and language modelling, while approximate inference can be performed efficiently.

The first model is parameterised by Hierarchical Pitman-Yor Process (HPYP) language models (Teh, 2006a), the second by feed-forward neural network language models (Bengio et al., 2003). These models offer contrasting solutions to the problem of sparsity inherent in long conditioning contexts, which accurate models require. The first uses carefully specified priors encoding the relative importance of elements in the conditioning context. The neural network models overcome sparsity by learning distributed representations of words, as well as arbitrary combinations of context elements through the neural network hidden layer. The advantage of the neural network approach is that we do not have to specify an ordering of the context elements.

Exact inference is infeasible as the marginal probability cannot be factored into a dynamic program due to the large conditioning contexts used. Therefore we propose approximate inference methods with particle filtering. We also propose an efficient decoding algorithm based on particle filtering that can adapt the beam size to the uncertainty in the model while jointly predicting POS tags and parse trees.

In addition to supervised parsing, we also perform semi-supervised learning to improve language modelling, and evaluate the syntactic language models qualitatively.

3.1 Generative transition-based parsing

The generative models we propose in this chapter are based on arc-standard transition-based parsing. The parser generates a word (and optionally its POS tag) when it is shifted onto the stack, similar to the model proposed by Cohen et al. (2011). Due to spurious ambiguity supervised models can be trained either with a deterministic oracle, approximating the tree probability with the probability of its oracle derivation, or with a non-deterministic oracle, by approximating the sum over possible derivations of the parse tree.

The joint probability distribution over a sentence with words $\mathbf{w}_{1:n}$, tags $\mathbf{t}_{1:n}$ and transition sequence $\mathbf{s}_{1:2n}$ is defined as

$$p(\mathbf{w}_{1:n}, \mathbf{t}_{1:n}, \mathbf{s}_{1:2n}) = \prod_{i=1}^{n} \Big(p(t_i|\mathbf{h}^t_{m_i}) p(w_i|t_i, \mathbf{h}^w_{m_i}) \prod_{j=m_i+1}^{m_{i+1}} p(s_j|\mathbf{h}^s_j) \Big), \tag{3.1}$$

where m_i is the number of transitions that have been performed just before (t_i, w_i) is shifted, and $\mathbf{h}^s_j$, $\mathbf{h}^t_j$ and $\mathbf{h}^w_j$ are the (bounded length) conditioning contexts at the jth transition for the transition, tag and word predictions, respectively. The transition and word (and tag) predictions are interleaved, as word predictions are triggered by shift actions, which are a subset of the full transition sequences (n actions out of length $2n$). Between the prediction of (t_i, w_i) and (t_{i+1}, w_{i+1}), transitions indexed m_i to m_{i+1} are performed. Shift actions occur at transitions s_{m_i}.

The first transition will always be shift, as the root node may not be removed from the stack. The generative process terminates when a right-arc is added from the root node to the head word, such that only the root remains on the stack. This implicitly generates the end-of-sentence token w_n.[1]

3.2 Inference

At every time step the transition system configuration is a function of the entire history of transition actions taken. Therefore, although we condition only on a fixed number of elements in the configuration, exact inference is intractable.

[1]We assume here that the root node has a single child, the head word.

The inference problems are to estimate the posterior distribution $p(\mathbf{s}_{1:2n}, \mathbf{t}_{1:n} | \mathbf{w}_{1:n})$, and the marginal distribution $p(\mathbf{w}_{1:n})$. For unsupervised learning we need to be able to sample from the posterior, while the marginal distribution is used for language modelling. For parsing we have to find the transition sequence that maximises the posterior. In addition to formulating the particle filter for sampling and estimating the marginal distribution sequentially, we propose a beam search decoder which is a heuristic approximation of the particle filtering algorithm.

3.2.1 Importance sampling

We view the model as a hidden state space model, where at each time step w_i is observed and the hidden state is a tuple

$$\mathbf{x}_i = (\mathbf{s}_{1:m_i}, \mathbf{t}_{1:i}) \tag{3.2}$$

representing the entire transition sequence history up to that point. The initial state $\mathbf{x}_0$ is an empty tuple.

The joint distribution then factorises as

$$p(\mathbf{w}_{1:n}, \mathbf{x}_{1:n}) = \prod_{i=1}^{n} p(\mathbf{x}_i | \mathbf{x}_{i-1}, \mathbf{w}_{1:i-1}) p(w_i | \mathbf{x}_i, \mathbf{w}_{1:i-1}), \tag{3.3}$$

where

$$p(\mathbf{x}_i | \mathbf{x}_{i-1}, \mathbf{w}_{1:i-1}) = \Big(\prod_{j=m_{i-1}+1}^{m_i} p(s_j | \mathbf{s}_{1:j-1}, \mathbf{t}_{1:i-1}, \mathbf{w}_{1:i-1}) \Big) p(t_i | \mathbf{s}_{1:m_i}, \mathbf{t}_{1:i-1}, \mathbf{w}_{1:i-1}), \tag{3.4}$$

which is approximated (with Markov assumptions) as

$$p(\mathbf{x}_i | \mathbf{x}_{i-1}, \mathbf{w}_{1:i-1}) \approx \Big(\prod_{j=m_{i-1}+1}^{m_i} p(s_j | \mathbf{h}^s_j) \Big) p(t_i | \mathbf{h}^t_{m_i}), \tag{3.5}$$

$$p(w_i | \mathbf{x}_i, \mathbf{w}_{1:i-1}) \approx p(w_i | t_i, \mathbf{h}^w_{m_i}). \tag{3.6}$$

Predicting x_i consists of a sequence of 0 or more reduce actions (left- or right-arcs) and a shift action which predicts the next tag and indicates that the next word should be predicted. At the final time step (where w_n is the end-of-sentence symbol), a sequence of reduce actions is performed that leaves only the root symbol remaining on the stack. The end-of-sentence tag and word prediction follow deterministically.

The state space grows exponentially over time; although we condition on a fixed number of elements at each time step, their choice is a function of the entire state space. Therefore exact inference with the forward-backward dynamic programming algorithm is not tractable. We first show how to perform approximate inference with importance sampling, and then introduce the particle filter to perform sequential importance sampling (Doucet and Johansen, 2009).

As we cannot sample efficiently from the posterior distribution $p(\mathbf{x}|\mathbf{w})$ we define a *proposal* distribution π from which we can sample efficiently:

$$\pi(\mathbf{x}|\mathbf{w}) = \prod_{i=1}^{n} p(\mathbf{x}_i|\mathbf{x}_{i-1}, \mathbf{w}_{1:i-1}). \tag{3.7}$$

This distribution can essentially be seen as a discriminative parsing model without look-ahead, as it is a product of the transition (and POS tag) probabilities. This proposal distribution is chosen as it enables sequential sampling (as defined below) and decomposes over time steps, while maintaining the expressiveness of the full model.

The *importance weights*, which are used to obtain an unbiased estimate of the marginal distribution by reweighing the (biased) samples from the proposal distribution, are defined as

$$\begin{aligned}
\omega(\mathbf{x}_{1:i}) &= \frac{p(\mathbf{x}_{1:i}, \mathbf{w}_{1:i})}{\pi(\mathbf{x}_{1:i}|\mathbf{w}_{1:i})} && (3.8)\\
&= \prod_{i=1}^{n} \frac{p(\mathbf{x}_i|\mathbf{x}_{i-1}, \mathbf{w}_{1:i-1})p(w_i|\mathbf{x}_i, \mathbf{w}_{1:i-1})}{p(\mathbf{x}_i|\mathbf{x}_{i-1}, \mathbf{w}_{1:i-1})} && (3.9)\\
&= \prod_{j=1}^{i} p(w_j|x_j, \mathbf{w}_{1:j-1}). && (3.10)
\end{aligned}$$

Suppose k independent samples are drawn from π:

$$\mathbf{x}^{(i)} \sim \pi(\mathbf{x}|\mathbf{w}) \qquad \text{for } i = 1, \ldots, k. \tag{3.11}$$

The empirical estimate of the posterior distribution is then

$$p(\mathbf{x}|\mathbf{w}) \approx \sum_{i=1}^{k} W^i \delta_{\mathbf{x}^{(i)}}(\mathbf{x}), \tag{3.12}$$

where δ is the Dirac delta function and

$$W_i = \frac{\omega(\mathbf{x}^{(i)})}{\sum_{j=1}^{k} \omega(\mathbf{x}^{(j)})}. \tag{3.13}$$

The Monte Carlo estimate of the marginal distribution is

$$p(\mathbf{x}) = \mathbb{E}_{\pi(\mathbf{x}|\mathbf{w})}\omega(\mathbf{x}) \tag{3.14}$$

$$\approx^{\text{MC}} \frac{1}{k}\sum_{i=1}^{k}\omega(\mathbf{x}^{(i)}). \tag{3.15}$$

3.2.2 Particle filtering

Sampling complete transition sequences from $\pi(\mathbf{x}|\mathbf{w})$ will have high variance; instead we perform *sequential* importance sampling. The posterior and marginal distributions satisfy the following recursions, which form the basis of particle filtering (Doucet and Johansen, 2009):

$$p(\mathbf{x}_{1:i}|\mathbf{w}_{1:i}) = p(\mathbf{x}_{1:i-1}|\mathbf{w}_{1:i-1})\frac{p(\mathbf{x}_i|\mathbf{x}_{i-1}, \mathbf{w}_{1:i-1})p(w_i|\mathbf{x}_i, \mathbf{w}_{1:i-1})}{p(w_i|\mathbf{w}_{1:i-1})}, \tag{3.16}$$

$$p(w_i|\mathbf{w}_{1:i-1}) = \sum_{\mathbf{x}_{i-1}} p(\mathbf{x}_{i-1}|\mathbf{w}_{1:i-1}) \sum_{\mathbf{x}_i} p(\mathbf{x}_i|\mathbf{x}_{i-1}, \mathbf{w}_{1:i-1})p(w_i|\mathbf{x}_i, \mathbf{w}_{1:i-1}). \tag{3.17}$$

The proposal distribution at time step i is

$$\pi(\mathbf{x}_i|\mathbf{w}_{1:i-1}) = p(\mathbf{x}_i|\mathbf{x}_{i-1}, \mathbf{w}_{1:i-1}). \tag{3.18}$$

In order to obtain unbiased samples with the particle filter, the importance weights can be estimated sequentially as

$$\omega(\mathbf{x}_{1:i}) = \prod_{j=1}^{i-1} p(w_j|x_j, \mathbf{w}_{1:j-1}) \cdot p(w_i|x_i, \mathbf{w}_{1:i-1}) \tag{3.19}$$

$$= \omega(\mathbf{x}_{1:i-1})p(w_i|x_i, \mathbf{w}_{1:i-1}). \tag{3.20}$$

The importance weights are normalised when sampling from the posterior (Equation 3.13), so the marginal $p(w_i|\mathbf{w}_{1:i-1})$ does not have to be computed explicitly.

Sequential importance sampling is used to obtain a Monte Carlo estimate of the marginal distribution,

$$p(w_i|\mathbf{w}_{1:i-1}) \approx^{\text{MC}} \frac{1}{k}\sum_{j=1}^{k}\omega(\mathbf{x}_{1:i-1}^{(j)}) \sum_{x_i} p(\mathbf{x}_i|\mathbf{x}_{i-1}^{(j)}, \mathbf{w}_{1:i-1})p(w_i|\mathbf{x}_i^{(j)}, \mathbf{w}_{1:i-1}), \tag{3.21}$$

where

$$\mathbf{x}_{i-1}^{(j)} \sim \pi(\mathbf{x}_{i-1}|\mathbf{x}_{i-2}^{(j)}, \mathbf{w}_{1:i-1}) \qquad \text{for } j = 1, \ldots, k. \tag{3.22}$$

In practice we shall also approximate the marginal distribution by summing over the beam after decoding with the method described in Section 3.2.3, showing that this is a low-variance estimate.

Now we define the bootstrap particle filter (Gordon et al., 1993) for sampling from the posterior. A set of k samples, called *particles*, are sampled sequentially from the proposal distribution. At every time step, each sample is extended by independently sampling transition actions s until s equals shift, and then sampling a tag t.

Over time the (normalised) importance weights might become very peaked, in practice assigning all the weight to a single particle. To address this, the bootstrap filter introduces a *selection* step:

At time step i we have samples from $\pi(\mathbf{x}_{1:i}|\mathbf{w}_{1:i})$ but the goal is to obtain samples from

$$p(\mathbf{x}_{1:i}|\mathbf{w}_{1:i}) = \omega(\mathbf{x}_{1:i})\pi(\mathbf{x}_{1:i}|\mathbf{w}_{1:i}). \tag{3.23}$$

These samples can be obtained by sampling from the distribution over the particles given by their (normalised) importance weights. This is referred to as *resampling*; k samples are selected as the new particles. The importance weights $\omega(\mathbf{x}_{1:i}^{(j)})$ of the resampled particles are reset to 1, as the samples are now unbiased.

In our implementation we resample at each time step, but alternatively one can set criteria to determine whether to resample or not at a give time step; the most common criterion is based on the effective sample size (Doucet and Johansen, 2009).

The complete particle filter as applied to our model is given in Algorithm 3. In the implementation we keep track of the count of each active particle rather than storing duplicate particles.

3.2.3 Decoding

Beam-search decoders for transition-based parsing (Zhang and Clark, 2008) keep a beam of partial derivations, advancing each derivation by one transition at a time. When the size of the beam exceeds a set threshold, the lowest-scoring derivations are removed. However, in an incremental generative model we need to compare derivations with the same number of words shifted, rather than same number of transitions performed (Titov and Henderson, 2007).

Algorithm 3 Particle filter for sampling derivations from the Markov generative dependency parser.

```
1: Given $\mathbf{w}_{1:n}$, number of particles $k$.
2: for $i = 1 \ldots n$ do
3:     Sampling step
4:     for $l = 1 \ldots k$ do
5:         $j \leftarrow m_{i-1}$
6:         while $s_j =$ Shift do
7:             $j \leftarrow j + 1$
8:             Sample $s_j^{(l)} \sim p(s|\mathbf{h}_j^{s(l)})$
9:         end while
10:        Sample $t_i^{(l)} \sim p(t|\mathbf{h}_j^{t(l)})$
11:        $\omega_i^{(l)} \leftarrow p(t_i^{(l)}|\mathbf{h}_j^{t(l)})p(w_i|t_i^{(l)}, \mathbf{h}_j^{w(l)})$
12:        $m_i \leftarrow j$
13:    end for
14:    Normalise $\omega_i$
15:    Selection step
16:    for $l = 1 \ldots k$ do
17:        Resample particles $\{(\mathbf{s}_{1:m_i}^{(l')}, \mathbf{t}_{1:i}^{(l')}), l' = 1, \ldots, k\}$ according to $\omega_i$
18:    end for
19: end for
20: return sampled derivations $\{(\mathbf{s}_{1:2n}^{(l)}, \mathbf{t}_{1:n}^{(l)}), l = 1, \ldots, k\}$.
```

One solution is to keep n separate beams, each containing only derivations with i words shifted, but this approach leads to $O(n^2)$ decoding complexity. For linear time decoding the total number of reduce transitions that can be performed over all derivations between two shift transitions has to be bounded.

We therefore propose a novel linear-time decoding algorithm which modifies the particle filter to become a beam-search heuristic (see Algorithm 4). The beam consists of partial derivations d_j. Instead of specifying a fixed limit on the size of the beam, the beam size is controlled by setting the number of particles k. Each d_j is associated with k_j particles, such that $\sum_j k_j = k$. First, instead of sampling, the procedure is made deterministic by dividing k_j proportionally between taking a shift or reduce transition, according to $p(s_j|\mathbf{h}_j^s)$. If a non-zero number of particles are assigned to reduce, the highest scoring left-arc or right-arc transitions are chosen deterministically, and derivations that execute them are added to the beam. In practice we found that adding only the highest scoring reduce transition gives very similar performance to adding multiple ones.

Algorithm 4 Beam search decoder for arc-standard generative dependency parsing.

```
1: Given Sentence $\mathbf{w}_{1:n}$, number of particles $k$.
2: *Initialize* the beam with parser configuration $d$ with weight $d.\theta = 1$ and $d.k = k$ particles
3: for $i = 1 \dots n$ do
4:     *Search step*
5:     for all derivation $d$ in beam do
6:         $n_{\text{shift}} = \text{round}(d.k \cdot p(\text{sh}|d.\mathbf{h}^s))$
7:         $n_{\text{reduce}} = d.k - n_{\text{shift}}$
8:         if $n_{\text{reduce}} > 0$ then
9:             $s = \text{argmax}_{s \neq \text{sh}}\, p(s|d.\mathbf{h}^s)$
10:            beam.append($dd \leftarrow d$)
11:            $dd.k \leftarrow n_{\text{reduce}}$
12:            $dd.\theta \leftarrow dd.\theta \cdot p(s|d.\mathbf{h}^s)$
13:            $dd$.execute($s$)
14:        end if
15:        $d.k \leftarrow n_{\text{shift}}$
16:        if $n_{\text{shift}} > 0$ then
17:            $d.\theta \leftarrow d.\theta \cdot p(\text{sh}|d.\mathbf{h}^s) \cdot \max_{t_i} p(t_i|d.\mathbf{h}^t)p(w_i|d.\mathbf{h}^w)$
18:            $d$.execute(sh)
19:        end if
20:    end for
21:    *Selection step*
22:    for all derivation $d$ in beam do
23:        $d.\theta' \leftarrow \frac{d.k \cdot d.\theta}{\sum_{d'} d'.k \cdot d'.\theta}$
24:        for all derivation $d$ in beam do
25:            $d.k = \lfloor d.\theta' \cdot k \rfloor$
26:            if $d.k = 0$ then beam.remove($d$)
27:            end if
28:        end for
29:    end for
30: end for
31: return Parse tree of $\text{argmax}_{d \text{ in beam}}\, d.\theta$.
```

The decoder can also perform POS tagging. Up to three candidate tags are assigned and corresponding derivations are added to the beam, with particles distributed relative to their tag probabilities (in Algorithm 4 only one tag is predicted).

One search step consists of iterating through the beam, including processing items added by reduce transitions during the pass, so that all items advance up to shifting the next word. The selection step reallocates the number of particles assigned to each beam item, again deterministically. Instead of computing importance weights, the reallocation is now based on the normalised weights of beam item derivations, each weighted by its current number of particles. The selection step allows the size of the beam to depend on the uncertainty of the model during decoding. The selectional branching method proposed by Choi and McCallum (2013) for discriminative beam-search parsing has a similar goal.

The final search step consists of performing reduce transitions on each derivation until they reach a terminal configuration. The dependency tree corresponding to the highest scoring final derivation is returned.

3.3 Probability models

3.3.1 Bayesian model

For the Bayesian model the word, tag and action distributions are estimated with hierarchical Pitman-Yor processes. Draws in the generative process are defined as

$$t|\mathbf{h}^t \sim T_{\mathbf{h}^t} \tag{3.24}$$

$$w|\mathbf{h}^w \sim W_{\mathbf{h}^w} \tag{3.25}$$

$$s|\mathbf{h}^s \sim S_{\mathbf{h}^s}, \tag{3.26}$$

where T, W and S are HPYPs for the tag, word and transition predictions, respectively. The HPYP models interpolate between different context sizes by dropping one element at each level in the HPYP to back off to a distribution lower in the hierarchy. The crucial modelling choice is therefore the selection and ordering of the context, in order to avoid sparsity in the back-off contexts. This choice can be seen as a hyperparameter of the model. In our model the context consists of words and tags of nodes that are on the stack or dependents of stack elements (see Table 3.1). For any node a, $lc_1(a)$ refers to the leftmost child node of a in the partially constructed dependency tree, and $rc_1(a)$ to its rightmost child. The second left-most and right-most child nodes are referred to as $lc_2(a)$ and $rc_2(a)$, respectively. The POS tag of a is referred to as $a.t$ and the word type as $a.w$.

The methodology followed to choose the contexts is described in Section 3.4 below. In the conditioning contexts, word types are always the first to be dropped in the back-off. The reason for this is that they are more sparse than POS tags, and the syntactic information conveyed by the tags is more informative in guiding the next parsing action. We shall also consider as a baseline unlexicalised models that are based only on POS tags.

Prediction	Context
a_i	$\sigma_1.t, \sigma_2.t, rc_1(\sigma_1).t, lc_1(\sigma_1).t, \sigma_3.t, rc_1(\sigma_2).t, \sigma_1.w, \sigma_2.w$
t_j	$\sigma_1.t, \sigma_2.t, rc_1(\sigma_1).t, lc_1(\sigma_1).t, \sigma_3.t, rc_1(\sigma_2).t, \sigma_1.w, \sigma_2.w$
w_j	$\beta.t, \sigma_1.t, rc_1(\sigma_1).t, lc_1(\sigma_1).t, \sigma_1.w, \sigma_2.w$

Table 3.1: HPYP conditioning contexts for the transition, tag and word distributions of the generative parser. The context elements are ordered from most important to least important; the last elements in the lists are dropped first in the back-off structure.

Training For supervised training, given a training set of parsed and tagged sentences, we use the oracle to extract a derivation (transition sequence) for each sentence. Training examples for the word, tag and transition prediction models are extracted from the derivations.

To perform unsupervised learning with the Bayesian model, particle filtering is combined with Gibbs sampling, giving rise to a particle Gibbs inference method (Andrieu et al., 2010). Initially we sample a derivation for each unlabelled training sentence, and add it to the Chinese Restaurant Process (CRP) model. Then for at Gibbs sampling iteration, for each sentence we remove the training examples from the current derivation of the sentence from the CRP, sample a new derivation, and add the examples for the new derivation back to the CRP. For semi-supervised learning we apply structural annealing, assuming that the distribution given by the labelled examples gives a good starting point. In the initial unsupervised iterations we use the Viterbi derivations of unlabelled sentences, only sampling from the distribution at later iterations (Spitkovsky et al., 2010b).

3.3.2 Neural network model

The second probability model is based on feed-forward neural networks. The word, tag and action distributions are estimated with neural networks that share input and hidden layers (and therefore use the same context), but with separate output layers.

The conditioning context templates are defined in Table 3.2. The neural network model allows us to include a large number of elements without suffering from sparsity, and in contrast to the HPYP model we do not have to specify any ordering of the elements.

Each context position is represented by vector representations of its word and POS tag; these representations form the input layer of the network. The network has a single hidden

Order	Elements
1	$\sigma_1, \sigma_2, \sigma_3, \sigma_4$
2	$lc_1(\sigma_1), rc_1(\sigma_1), lc_1(\sigma_2), rc_1(\sigma_2), lc_2(\sigma_1), rc_2(\sigma_1), lc_2(\sigma_2), rc_2(\sigma_2)$
3	$lc_1(lc_1(\sigma_1)),\ rc_1(rc_1(\sigma_1)),\ lc_1(lc_1(\sigma_2)),\ rc_1(rc_1(\sigma_2))$

Table 3.2: Conditioning context elements for the feed-forward neural network: First, second and third order dependencies are used.

layer, and we experiment with different non-linearities. The word probability distribution is estimated with a class-factored softmax.

The supervised model is trained to maximise the joint distribution over parsed training sentences. For our experiments we train the model while the training objective improves and choose the parameters of the iteration with the best development set accuracy, performing early stopping. The model obtains high accuracy with only a few training iterations.

3.4 Experiments

3.4.1 Data

We evaluate our model as a parser on the widely used English Penn treebank (Marcus et al., 1993) setup, training on WSJ sections 02-21, developing on section 22, and testing on section 23. We use both the YM and SD representations (see section 2.2.2). We report unlabelled attachment score (UAS) and labelled attachment score (LAS), excluding punctuation.

Unknown words are replaced by tokens representing morphological surface features (based on capitalization, numbers, punctuation and common suffixes) similar to those used in the implementation of generative constituency parsers (Klein and Manning, 2003). In particular we reimplement the unknown word classifier of the Berkeley parser.[2] Words that occur only once in the training data are replaced by their unknown word classes.

3.4.2 Model setup

Our HPYP dependency parser (referred to as **HPYP GenDP** when reporting results) is trained with 20 iterations of Gibbs sampling, resampling the hyperparameters after every

[2]http://github.com/slavpetrov/berkeleyparser

Model	UAS	LAS
MaltParser unlexicalised	85.23	82.80
MaltParser lexicalised	89.17	87.81
Unlexicalised	85.64	82.93
Lexicalised, unlex context	87.95	85.04
Lexicalised, tagger POS	87.84	85.54
Lexicalised, predict POS	**89.09**	**86.78**
Lexicalised, gold POS	89.30	87.28

Table 3.3: HPYP parsing accuracies on the YM development set, for various lexicalised and unlexicalised setups.

iteration. Training with a deterministic oracle takes 28 seconds per iteration (excluding re-sampling hyper-parameters), while a non-deterministic oracle (sampling with 100 particles) takes 458 seconds.

The implementation of our neural network parser (**FFNN GenDP**) is partly based on the C++ `OxLM` neural language modelling framework (Baltescu et al., 2014).[3] The model parameters are initialised randomly by drawing from a Gaussian distribution with mean 0 and variance 0.1, except for the bias weights, which are initialised by the unigram distributions of their output. We use mini-batches of size 128, L2 regularisation parameter 10, and word representation and hidden layer sizes 256. The AdaGrad learning rate is initialised to 0.05.

3.4.3 Bayesian modelling choices

We consider several modelling choices in the construction of the Bayesian model. The model is trained on the YM dataset, and we evaluate different modelling choices on the YM development set. Development set parsing accuracies are given in Table 3.3.

As a discriminative baseline we use MaltParser (Nivre et al., 2006), a discriminative arc-standard parser with a linear SVM classifier. Although the accuracy of this model is not state-of-the-art, it does enable us to compare against a discriminative model with a feature set based on the same elements as in our conditioning contexts.

[3]Code for both the Bayesian and neural models are available at https://github.com/janmbuys/oxdp.

Context	UAS	LAS
$\sigma_1.t, \sigma_2.t$	73.25	70.14
$+rc_1(\sigma_1).t$	80.21	76.64
$+lc_1(\sigma_1).t$	85.18	82.03
$+\sigma_3.t$	87.23	84.26
$+rc_1(\sigma_2).t$	87.95	85.04
$+\sigma_1.w$	88.53	86.11
$+\sigma_2.w$	88.93	86.57

Table 3.4: Parsing accuracy for various sized HPYP conditioning contexts. Results are given on the YM development set.

HPYP priors The first modelling choice is the selection and ordering of elements in the conditioning contexts of the HPYP priors. Table 3.4 shows how the development set accuracy increases as more elements are added to the conditioning context. The results are for jointly predicting POS tags (see below). The first two words on the stack are the most important, but insufficient – second-order dependencies and further stack elements should also be included in the context. The challenge is that the back-off structure of each HPYP specifies an ordering of the elements based on their importance in the prediction. We are therefore much more restricted than classifiers with large, sparse feature-sets which are commonly used in transition-based parsers. Due to sparsity, the word types are the first elements to be dropped in the back-off structure, and elements such as third-order dependencies cannot be included successfully in our model.

Sampling over parsing derivations during training improves performance marginally by 0.16% to 89.09 UAS. When using a single unknown word token instead of multiple unknown word classes, performance drops to 88.60 UAS.

POS tagging The standard practice in transition-based parsing is to obtain POS tags with a stand-alone tagger before parsing. However, as our generative model includes a distribution over POS tags, we can use the model to assign POS tags jointly during decoding. We compare predicting tags against using gold standard POS tags and tags obtained using the Stanford POS tagger (Toutanova et al., 2003). For the Stanford tagger we use the efficient "left 3 words" model, trained on the same data as the parsing model, excluding distributional features. Tagging accuracy is 95.9% on the development set and 96.5% on the test

Algorithm	Beam size	Sentences/sec	UAS
Beam	32	3	88.82
Beam	16	7	88.46
Particle	5000	18	89.03
Particle	1000	27	88.93
Particle	100	54	87.99
Particle	10	104	85.27

Table 3.5: Speed-accuracy trade-offs for HPYP parsing different decoding configurations (beam search or particle filtering), jointly predicting POS tags.

Algorithm	Beam size	Sentences/sec	UAS
Beam	16	29	87.64
Particle	1000	108	87.59
Particle	100	198	87.46
Particle	10	333	85.86

Table 3.6: Speed-accuracy trade-offs for HPYP parsing different decoding configurations (beam search or particle filtering), separate POS tagging.

set. Even though the tags predicted by our model are slightly less accurate than the Stanford tags on the development set (95.6%), jointly predicting tags and decoding increases the UAS by 1.1% (see Table 3.3). The joint prediction is a better fit to the generative model. However, using gold POS tags still gives the best parsing accuracy.

Lexicalisation We train lexicalised and unlexicalised versions of our model (see Table 3.3). Unlexicalised parsing gives us a strong baseline over which to consider our model's ability to predict and condition on words. Unlexicalised parsing is also considered to be robust for applications such as cross-lingual parsing (McDonald et al., 2011). Additionally we consider a version of the model that does not include lexical elements in the conditioning context, satisfying a HMM-like factorisation. This model performs only 1% UAS lower than the best lexicalised model, although it makes much stronger independence assumptions.

Speed-accuracy trade-offs We consider the trade-off between speed and accuracy in the model, which can be controlled through the choice of decoding algorithm, beam size (or

Model	UAS	LAS
Words only, 3rd order	89.30	87.78
Word and tags, 2nd order	90.34	89.08
Words and tags, 3rd order	90.52	89.29

Table 3.7: Feed-forward neural network parsing accuracies on the YM development set.

number of particles) and predicting POS tags either jointly or separately. The results are in Tables 3.5 and 3.6.

The optimal number of particles is found to be 1000 - more particles only increase accuracy by about 0.1 UAS. Although jointly predicting tags is more accurate, models using pre-obtained tags are more accurate when comparing faster models: At a speed of approximately 100 sentences per second the pre-tagged model's accuracy is 87.59 UAS (Table 3.6) against 85.27 UAS for the joint model (Table 3.5). In comparison, the MaltParser parses approximately 375 sentences per second.

We also compare our particle filter-based algorithm against a more standard beam-search algorithm that prunes the beam to a fixed size after each word is shifted. This algorithm is much slower than the particle-based algorithm – to get similar accuracy it parses only 3 sentences per second (against 27) when predicting tags jointly, and 29 (against 108) when using pre-obtained tags.

3.4.4 Neural network modelling choices

Development set results for the neural network model are given in Table 3.7, based on the YM representation. Third order dependencies improves performance by 0.18% UAS over models based on first and second order dependencies. Including additional elements beyond our third-order feature set (for example children of the third stack item) did not improve performance further in our experiments. Modelling POS tags (but using separately predicted tags) improved performance by 1.2% UAS and 1.5% LAS.[4] This shows that the distributed representations learned by the network are able to capture most of the lexical categorization of the syntactic role of words, as opposed to non-distributed models which

[4]Joint tagging is prohibitively slow.

Model	UAS	LAS	Sentences/sec
Wallach et al. (2008)	85.7	-	-
Titov and Henderson (2007)	90.75	89.29	1
MaltParser	88.88	87.41	375
Zhang and Nivre (2011)	92.9	91.8	29
Choi and McCallum (2013)	92.96	91.93	110
Martins et al. (2013)	93.07	-	42
HPYP GenDP	**88.47**	**86.13**	18
FFNN GenDP	**90.16**	**88.83**	4

Table 3.8: Parsing accuracies on the YM test set, compared with published results. Decoding speeds are also given. Titov and Henderson (2007) was retrained to enable direct comparison.

rely much more on POS. For the choice of activation function of the neural network hidden layer we found that the sigmoid function gives the best performance, following by `tanh`.

3.4.5 Comparative parsing results

Test set results comparing our models against existing discriminative and generative dependency parsers are given in Tables 3.8 and 3.9.

Our HPYP model performs much better than the Bayesian version of Eisner's generative model (Wallach et al., 2008). The accuracy of this model is only 2.3 UAS below the generative model of Titov and Henderson (2007), despite that model being much more powerful. The UAS of our model is very close to that of the MaltParser. However performance is relatively worse on LAS than on UAS. An explanation for this is that as we do not include labels in the conditioning contexts, the predicted labels are independent of words that have not yet been generated. Despite these promising results, the HPYP model's performance still lags behind discriminative parsers with beam-search and richer feature sets (Zhang and Nivre, 2011; Choi and McCallum, 2013), as well as the third-order graph-based parser of Martins et al. (2013).

Our neural generative model outperforms the Bayesian model by a large margin. The model's accuracy is 0.6% UAS below the generative model of Titov and Henderson (2007) on the YM test set, despite that model being able to condition on arbitrary long contexts. Our model is also much faster than Titov and Henderson's model, for both decoding and

Model	UAS	LAS
Titov and Henderson (2007)	90.93	89.42
FFNN GenDP	**91.11**	**89.41**

Table 3.9: Parsing accuracies on the WSJ test set, CoNLL dependencies.

Model	UAS	LAS
MaltParser	88.9	86.2
Chen and Manning (2014)	91.8	89.6
Dyer et al. (2015)	93.1	90.9
Weiss et al. (2015)	93.99	92.05
Titov and Henderson (2007)	91.43	89.02
HPYP GenDP	87.9	86.2
FFNN GenDP	**90.10**	**87.74**

Table 3.10: Dependency parsing accuracies on the WSJ test set, Stanford dependencies.

training (the Titov and Henderson model takes 3 days to train). On the CoNLL dependencies dataset our model is slightly more accurate than Titov and Henderson's model (Table 3.9).

We also evaluate our models on Stanford dependencies to enable comparison against recent neural network dependency parsers (Table 3.10). Both our models are less accurate on this representation, a tendency that has widely been observed in discriminative parsing, as the Stanford dependencies represent more complex relations.

Analysis We argue that the main weakness of the HPYP parser is sparsity in large conditioning contexts composed of tags and words. The POS tags in the parser configuration context already give a very strong signal for predicting the next transition. As a result it is challenging to construct PYP reduction lists that also include word types without making the back-off contexts too sparse. The generative neural network model overcomes those limitations, but still underperforms compared to discriminative models.

The other limitation is that our decoding algorithm, although efficient, still prunes the search space aggressively, while not being able to take advantage of look-ahead features as discriminative models can. We note that discriminative parsers cannot attain high performance without look-ahead features, even when a large beam is used (see Section 4.4).

Model	Dev perplexity	Test perplexity
IKN 5-gram	155.93	170.09
HPYP 5-gram	156.61	171.41
FFNN 5-gram	147.80	157.13
HPYP GenDP (YM)	157.35	168.36
HPYP GenDP (SD)	147.47	162.04
FFNN GenDP (YM)	152.54	159.34
FFNN GenDP (YM) + unsup	139.62	148.33
FFNN GenDP (SD)	138.96	149.69
FFNN GenDP (SD) + unsup	126.79	138.62

Table 3.11: WSJ language modelling development and test results. We compare our models, with and without unsupervised tuning, to n-gram baselines.

3.4.6 Language modelling

Next we evaluate our models as syntactic language models, first on the WSJ and then on larger unlabelled corpora with semi-supervised learning. We perform unlabelled parsing, as the labels are not included in the conditioning contexts to make transition and word predictions. For the HPYP model we use jointly predicted POS tags, but for the neural model the POS prediction cost outweighs the small potential gain.

Language models are evaluated intrinsically by estimating *perplexity* on held-out datasets. Cross-entropy is the average negative log likelihood per symbol of the test data according to the model.[5] Perplexity is cross-entropy exponentiated.

Perplexity results on the WSJ are given in Table 3.11, using the same vocabulary as for parsing. As baselines we report results on interpolated Kneser-Ney (IKN) (Kneser and Ney, 1995), HPYP and neural network 5-gram models (using a larger context results in very limited perplexity gains). The small difference between the Kneser-Ney and Bayesian models can be ascribed to implementation differences. The neural n-gram model gives better performance than both the count-based n-gram and syntactic models.

For our dependency-based language models we report perplexities computed by summing over the final beam in the particle filter-based decoder, as it is intractable to sum over all possible parses to compute the true marginal probability. This gives an upper bound

[5]We exclude end-of-sentence symbols from the symbol count.

on the true model perplexity (lower perplexities are better). Increasing the number of particles has a very small impact on perplexity, which indicates that the bound is relatively small. Alternatively a Monte Carlo estimate can be obtained with the particle filter, based on Equation 3.21. In addition to the syntactic features used for parsing, the neural syntactic language model also uses the 5-gram word context in its input layer.

First we train the models with supervised parse trees. Both models perform considerably better with the Stanford dependencies. While the supervised YM models perform very similar to n-gram models of the same type, the Stanford dependencies yield perplexity improvements on both the HPYP and FFNN models.

Second we consider a training setup where we first perform 5 supervised iterations, and then continue to train without supervision, treating the transition sequence as latent. Parse trees are sampled with a particle filter. Updates are performed for every sentence for the Bayesian model, and for every mini-batch for the neural network model. For the Bayesian model this is an instance of particle Gibbs sampling (Andrieu et al., 2010). This approach further improves the perplexities of the neural models, but the SD-initialised model still gives the best performance due to SGD's sensitivity to initial conditions, yielding a 18.5% perplexity reduction relative to the Kneser-Ney model. We did not observe significant improvements with the Bayesian model following this approach.

While trained on the same sentences, this approach allows the model to learn parses that are not necessarily consistent with the annotated parse trees. The unsupervised training stage results in the parsing accuracy dropping by about 2%. We postulate that the model is learning to make small adjustments to favour parses that explain the data better than the annotated parse trees, leading to the improvement in perplexity.

Semi-supervised training We consider a semi-supervised setup where we train on large unsupervised corpora from the WMT News Crawl Data.[6] The models are evaluated on the `newstest2012` test set. We used two different subsets: 24.1 million words for the Bayesian model and 7 million words for the neural model. The vocabularies were obtained based on the available training data, so the results of the two setups are not directly comparable.

[6]Available at http://www.statmt.org/wmt14/translation-task.html.

Model	Perplexity
HPYP 5-gram	178.13
HPYP GenDP	**163.96**
FFNN 5-gram	203.5
FFNN GenDP	**200.7**

Table 3.12: Semi-supervised dependency language modelling, testing on `newstest2012`. The HPYP and neural models were trained on different datasets.

Semi-supervised learning is performed as follows: After training the model on the WSJ we parsed the unannotated data with the model, and continued to train on the highest-scoring parses. This is similar to Viterbi EM training for unsupervised learning, or self-training for parsing (McClosky et al., 2006). For the HPYP model only the word prediction distribution is updated, not the tag and transition distributions. An alternative approach would be to use an external discriminative parser to parse the unlabelled data, and train the generative model on that. External parsers have been shown to improve semi-supervised learning, for example using tri-training (Zhou and Li, 2005).

We observe improvements in perplexity for both the Bayesian and neural models (Table 3.12). This is a promising result, as it shows that our model can successfully generalise to larger vocabularies and unannotated datasets. We expect larger improvements when training on more data and with more sophisticated inference.

3.4.7 Generation

To evaluate our generative model qualitatively we performed unconstrained generation of sentences (and parse trees) from the model. We found that sentences generated by our syntactic generative models display a higher degree of syntactic coherence, while retaining the local coherence of the n-gram models. Example sentences are given in Table 3.13. For each model we generated 1000 (unbiased) samples and selected the 10 highest-scoring sentences of length 10 or more. The model vocabularies include unknown word classes.

The quality of the sentences generated by the dependency model is superior to that of the n-gram model, despite the models having similar test set perplexities. The dependency model was also able to generate balanced quotation marks.

otherwise , actual profit is compared with the 300-day estimate .
the companies are followed by at least three analysts , and had a minimum five-cent change in actual earnings .
bonds : shearson lehman hutton treasury index NUM , up
posted yields on NUM year mortgage commitments for delivery within NUM days .
in composite trading on the new york mercantile exchange .
the company , which has NUM million shares outstanding .
the NUM results included a one-time gain of \$ NUM million .
however , operating profit fell NUM % to \$ NUM billion from \$ NUM billion .
merrill lynch ready assets trust : NUM % NUM days ; NUM % NUM to NUM days ;
in new york stock exchange composite trading , one trader .
sales rose NUM to NUM million from \$ NUM .
estimated volume was about \$ NUM a share , .
meanwhile , annual sales rose to NUM % from \$ NUM .
mr. bush 's profit climbed NUM % , to \$ NUM from \$ NUM million million , or NUM cents a share .
treasury securities inc. is a unit of great issues .
“ he is looking out their shareholders , ” says .
while he has done well , she was out .
that 's increased in the second quarter 's new conventional wisdom .
mci communications said net dropped NUM % for an investor .
association motorola inc. , offering of \$ NUM and NUM cents a share .
the u.s. union board said revenue rose 11 % to \$ NUM million , or \$ NUM a share .
mr. bush has UNK-ed a plan to buy the company for \$ NUM to NUM million , or \$ NUM a share .
the plan was UNK-ed by the board 's decision to sell its \$ NUM million UNK loan loan funds .
in stocks coming months , china 's NUM shares rose 10 cents to \$ NUM million , or \$ NUM a share .
in the case , mr. bush said it will sell the company business UNK concern to buy the company .
it was NUM common shares in addition , with \$ NUM million , or \$ NUM a share , according to mr. bush .
in the first quarter , 1989 shares closed yesterday at \$ NUM , mr. bush has increased the plan .
last year 's retrenchment price index index rose 11 cents to \$ NUM million , or \$ NUM million is asked .
last year earlier , net income rose 11 million % to \$ NUM million , or 91 cents a share .
the u.s. union has UNK-ed \$ NUM million , or 22 cents a share , in 1990 , payable nov. 9 .

Table 3.13: Sentences sampled from the generative models: (above) IKN n-gram model; (middel) HPYP-DP; (bottom) FNN-DP.

3.5 Conclusion

We presented an approach to generative dependency parsing models that, unlike previous models, retains most of the speed and accuracy of discriminative parsers. Our models can accurately estimate probabilities conditioned on long context sequences. They scale to large training and test sets and, even though joint probability distribution over sentences and parses have to be estimated, decoding is efficient. Additionally, the syntactic generative models give strong language modelling performance. For future work we believe that these models can be applied successfully to natural language generation tasks such as machine translation.

Chapter 4

Recurrent Models for Generative Dependency Parsing

Recent advances in language modelling have been driven by the success of recurrent neural networks (RNNs), in particular Long Short-term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) RNNs and their variants. RNNs are able to model long-distance dependencies that could not be captured by count-based or neural n-gram language models (Kneser and Ney, 1995; Bengio et al., 2003). While it has been shown that RNNs can model hierarchical bracketing structure (Karpathy et al., 2015), their ability to learn context-free grammars or more expressive formal languages has been questioned (Grefenstette et al., 2015). It has also been shown that LSTMs can predict long-distance, syntax-sensitive dependencies only when trained with explicit supervision (Linzen et al., 2016).

In this chapter we extend the approach to generative dependency parsing proposed in Chapter 3 to models based on RNNs. A complementary goal is to formulate incremental generative parsers (that can act as syntactic language models) which are amenable to exact inference with dynamic programming. By treating the syntax (expressed by transition system derivations) as a latent variable, dynamic programming enables unsupervised learning, exact marginalisation for language modelling and exact decoding for parsing. We perform both supervised and unsupervised learning, applying both the supervised and unsupervised models to language modelling.

In particular we propose generative models based on the arc-eager and arc-hybrid transition systems with $O(n^4)$ dynamic programs based on Kuhlmann et al. (2011). While the conditioning contexts licensed by these dynamic programs were too restrictive for traditional models, recent work has shown that using sequential RNNs to learn underlying features enables accurate parsing with very small features sets (Kiperwasser and Goldberg, 2016; Cross and Huang, 2016).

Our GPU implementation enables both exact decoding which is feasible in practice, and unsupervised learning with backpropagation through the dynamic program to the RNN hidden states (which encode sequential context).

4.1 Background

We review a number of recent approaches to augment RNNs to enrich their expressiveness. Neural abstract machines augment neural networks (usually RNNs) with external memory, including stacks and other data structures, that are operated on with differentiable operations to enable end-to-end learning. Bahdanau et al. (2015) proposed an attention mechanism for encoder-decoder RNNs: At each time step in the decoder, an input representation is computed by taking the expectation over the alignment between the current RNN state and all the encoder states. This representation is then used to make predictions with the decoder, and also fed into the decoder RNN to inform subsequent predictions. The alignment is not treated as a latent variable; instead what is called a *soft* attention mechanism is learned end-to-end by backpropagating from the predicted decoder outputs through the expectation over alignments.

Memory networks (Weston et al., 2014; Sukhbaatar et al., 2015) use a similar mechanism to read from a memory of RNN states. Neural Turing machines (Graves et al., 2014) have read-write memory that is updated at each time step. Grefenstette et al. (2015) proposed a neural stack that is operated on with differentiable push and pop computations for end-to-end training. The stack representation is continuous. Every entry has an associated real number indicating its strength; push and pop operations update these values rather than executing discrete actions. On synthetic tasks these models outperform RNNs in learning sequential structure equivalent to context-free grammars and more expressive formal languages.

The advantage of the models we propose over neural abstract machines is that we learn probability distributions over parse trees or stack configurations. These distributions are interpretable, can be augmented with priors, and the discrete structures learned can be applied to inference and reasoning.

Reinforcement learning has been proposed to learn compositional tree-based representations in the context of an end task (Andreas et al., 2016; Yogatama et al., 2016), but this approach has high variance and provides no guarantees of finding optimal trees.

More recent research has proposed neural models with structured latent variables: Rastogi et al. (2016) incorporated neural context with a bidirectional RNN into weighted finite-state transducers, while Tran et al. (2016) proposed a neural hidden Markov model for Part-of-Speech (POS) induction. Yu et al. (2016) proposed a neural transduction model with polynomial-time inference where the alignment is a latent variable. Finally Kim et al. (2017) proposed structured attention mechanisms that compute features by taking expectations over latent structure. They define a tree-structured model with a latent variable for head selection, along with projectivity constraints. The model learns soft head selection, which is used as a feature in an attention-based decoder.

There has been a number of proposals for generative transition-based models based on RNNs. The architecture of Titov and Henderson (2007) is a recurrent model with additional connections to previous recurrent states at positions determined by the parser configuration. However, the model was not applied to language modelling, and the complexity of the architecture slows it down significantly. Dyer et al. (2016) proposed a generative transition-based RNN constituency parser that is accurate for both parsing and language modelling. The stack is encoded with a stack LSTM (Dyer et al., 2015) and parse trees are generated top-down. Both these models are based on unbounded contexts which require approximate inference to compute marginal sentence probabilities, and unsupervised learning is intractable.

Previous work have proposed dynamic programming algorithms that allow exact inference for transition-based parsers (Huang and Sagae, 2010; Kuhlmann et al., 2011; Cohen et al., 2011). However these algorithms either required approximate inference due to a too high polynomial order run-time complexity (Huang and Sagae, 2010), or had too restrictive feature spaces to be used as accurate models (Kuhlmann et al., 2011; Cohen et al., 2011). In concurrent work, Shi et al. (2017) proposed accurate discriminative parsers with

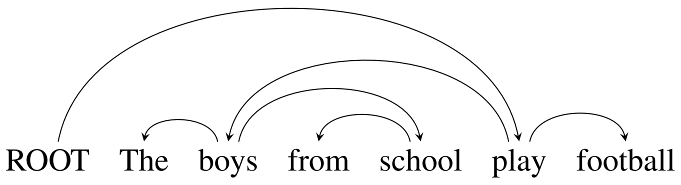


Figure 4.1: An unlabelled dependency tree.

minimal feature sets based on bidirectional LSTMs in which exact decoding and globally-normalised discriminative training is tractable with dynamic programming.

4.2 Shift-reduce parsing

We start by defining a shift-reduce transition system in which the transition actions do not add any dependency arcs, but simply process the words in the sentences left to right through shifting words on to a stack and reducing (popping) them from the stack. We define a generative model for this transition system and a dynamic program to perform inference over all possible shift-reduce transitions to process a given sentence. In the next section we extend this approach to transition systems for dependency parsing that are also based on shift-reduce actions.

As defined in Section 2.3, the state variables of the transition system are the stack σ and buffer index β. The initial configuration is $(\sigma, \beta) = ([0], 1)$. We define two transition actions, *shift* and *reduce*. Shift updates the transition state from (σ, j) to $(\sigma|j, j{+}1)$. Reduce changes the state from $(\sigma|i, j)$ to (σ, j).

When processing $\mathbf{w}_{1:n}$, if $\beta < n$ the root node may not be reduced from the stack, but if $\beta = n$ the system has to reduce. The terminal configuration is $(\emptyset, n)$. The root node is allowed to have multiple dependents, as the dynamic program we propose below is unable to enforce a single head word constraint. An example transition sequence for the sentence in Figure 4.1 is given in Table 4.1, along with the sequences for the transition systems to be defined in Section 4.3.

Transition configuration		Next Action		
Stack σ	Buffer β	SR	AH	AE
ROOT	The	sh	sh	sh
ROOT, The	boys	re	la	la
ROOT	boys	sh	sh	sh
ROOT, boys	from	sh	sh	sh
ROOT, boys, from	school	re	la	la
ROOT, boys	school	sh	sh	ra
ROOT, boys, school	play	re	ra	re
ROOT, boys	play	re	la	la
ROOT	play	sh	sh	ra
ROOT, play	football	sh	sh	ra
ROOT, play, football	EOS	re	ra	re
ROOT, play	EOS	re	ra	re
ROOT	EOS	re	la	la

Table 4.1: Transition system derivation for the sentence "The boys from school play football." The actions for the shift-reduce (SR), arc-hybrid (AH) and arc-eager (AE) transition systems are given. The transition actions are shift (sh), reduce (re), left-arc (la) and right-arc (ra). For clarity words rather than indexes are used to represent the parsing state. At each step the stack and buffer is shown, as well as the next transition action for each transition system. The different transition systems share the same stack configuration at each step.

Stack σ	Index $\beta - 1$	Prediction
ROOT	ROOT	sh(The)
ROOT, The	The	re
ROOT	The	sh(boys)
ROOT, boys	boys	sh(from)
ROOT, boys, from	from	re
ROOT, boys	from	sh(school)
ROOT, boys, school	school	re
ROOT, boys	school	re
ROOT	school	sh(play)
ROOT, play	play	sh(football)
ROOT, play, football	football	re
ROOT, play	football	re
ROOT	football	re

Table 4.2: Shift-reduce derivation with the *stack-next* generative formulation.

4.2.1 Generative model

The generative model for this transition system defines a joint probability distribution $p(\mathbf{w}_{1:n}, \mathbf{s}_{1:2n})$.[1] A shift action in the transition system generates (assigns probability to) the next word in the sentence. We consider two generative processes: In the first, shift generates $w_{\beta+1}$, the next word on the buffer. This is referred to as *buffer-next*. Alternatively shift generates w_β, which is being shifted from the buffer to the stack. This is referred to as *stack-next*. For clarity we shall formulate the models below with *buffer-next* but we also perform experiments with *stack-next*.

The second formulation has a more intuitive generative story as the generation of a word is conditioned on the top of the stack when it is generated (see Table 4.2), but the first formulation has the advantage that transition predictions are conditioned on the current word at position β, which is more informative for parsing predictions. The terminal conditions for two the generative processes are also different: The buffer-next model predicts the end-of-sentence token explicitly, after which reduce transitions are performed deterministically. The stack-next model does not predict an end-of-sentence token, instead terminating when the stack is empty.

The probability model is based on an RNN which encodes $\mathbf{w}$ left-to-right similar to

[1]We do not model POS tags here.

Action	State before	State after	Probability
Shift	$(S\|i, j)$	$(S\|i\|j, j+1)$	$p_{act}(\mathrm{sh}\|\boldsymbol{h}_i, \boldsymbol{h}_j) p_{gen}(w_{j+1}\|\boldsymbol{h}_i, \boldsymbol{h}_j)$
Reduce	$(S\|l\|i, j)$	$(S\|l, j)$	$p_{act}(\mathrm{re}\|\boldsymbol{h}_i, \boldsymbol{h}_j)$

Table 4.3: The shift-reduce transition system.

RNN language models (Section 2.1.3). The representation is computed independently of s. We use LSTM cells (Hochreiter and Schmidhuber, 1997) for the RNN. Let $\boldsymbol{h}_{0:n}$ be the RNN encoding, where $\boldsymbol{h}_i$ is the RNN hidden state after $\mathbf{w}_{1:i}$ has been encoded[2], therefore representing w_i and its left context. The stack at transition j is denoted as $\sigma^{(j)}$.

The generative model factors as

$$p(\mathbf{w}_{1:n}, \mathbf{s}_{1:2n}) = \prod_{i=1}^{n} \Big(p(w_i | \boldsymbol{h}_{\sigma_1^{(m_i)}}, \boldsymbol{h}_{i-1}) \prod_{j=m_i+1}^{m_{i+1}} p(s_j | \boldsymbol{h}_{\sigma_1^{(j)}}, \boldsymbol{h}_i) \Big), \tag{4.1}$$

where m_i is the number of transitions that have been performed when (t_i, w_i) is shifted. The overall structure is similar to that of the model in section 3.1: Between the current and next word predictions a number of transitions are predicted, conditioned on the last RNN hidden state as well as the RNN state corresponding to the top of the stack. The transition system and its parameterisation is given in Table 4.3. The final transition of a derivation reduces the root symbol from the stack.

The transition and word probability distributions are estimated by non-linear output layers that take context representations of positions in the transition system as input:

$$p_{\text{act}} = \text{sigmoid}(r^T \text{relu}(W_{as} \boldsymbol{h}_{\sigma_1} + W_{tb} h_\beta)) \tag{4.2}$$

$$p_{\text{gen}} = \text{softmax}(R^T \tanh(W_{gs} \boldsymbol{h}_{\sigma_1} + W_{gb} h_\beta)) \tag{4.3}$$

The model has two ways of representing context: The RNN encoding, which has a recency bias, and the stack, which can represent long range dependencies and has a syntactic distance bias. The choice of RNN states corresponding to stack elements to condition on is restricted by our goal of making the dynamic programming tractable. The choice of which stack elements to condition on is restricted in order to make dynamic programming tractable.

[2]The start of sentence symbol is encoded at position 0.

4.2.2 Dynamic program

The dynamic program we define is based on the dynamic programs proposed by Kuhlmann et al. (2011) and their application to generative dependency parsing (Cohen et al., 2011). Although there is an exponential number of transition sequences to process a given sequence, under certain assumptions of what conditioning information is available to make predictions, computations can be shared to enable polynomial-time parsing algorithms and polynomial-space representations of parse forests.

The key to the dynamic program is the decomposition of the transition sequence into *push computations*. A push computation is a sequence of transitions $c_0, \ldots, c_m$ which results in a single node pushed to the stack: The initial stack is not modified during the computation.

The simplest push computation is a single shift operation. Push computations can be composed recursively: Two consecutive push computations followed by a reduce transition yields a new push operation. Similarly a push computation can be decomposed into two sub-computations and a reduce operation.

A deduction system (Shieber et al., 1995) is then constructed to tabulate the computations. Items in the deduction system are based on the tuple of word indexes (i, j), which has the interpretation that there exists a push computation between actions a_k and a_l such that $\beta = i$ at time step k and $\sigma_1 = i$ and $\beta = j$ at time step l. In the derivation of the dynamic program (Kuhlmann et al., 2011), an additional index h is tracked, such that $\sigma_1 = h$ at time time step l. However, it is shown that $i = h$ for all derivable items in the transition systems we use.

To compute probabilities inside the dynamic program we need access to the feature space (conditioning context) of predictions corresponding to items that are derived. More specifically, the deduction system items have to be extended to keep track of the feature space indexes of the configurations before and after a push computation is performed, so that the probabilities associated with inference rules can be computed. Our model's feature space is (σ_1, β), so deduction system items are defined to have the form $[l, i, j]$, where (l, i) defines the feature space of the configuration before the push operation and (i, j) the feature space afterwards. This extends the items defined by Kuhlmann et al. (2011).

The deduction system is then defined as follows ($\emptyset$ denotes an empty element, which is used to refer to the second position on the stack in case there is only one element on the stack):

Goal:

$$[\emptyset, 0, n]$$

Axiom:

$$[\emptyset, 0, 1]$$

Deduction rules:

$$[l, i, j-1] \rightarrow [i, j-1, j] \qquad \text{(shift)}$$
$$[l, i, k][i, k, j] \rightarrow [l, i, j] \qquad \text{(reduce)}$$

The deduction (inference) rules are used to derive new items from existing ones, similar to the way that push computations are constructed: Performing shift on the state resulting from any existing push computation constitutes a new push computation that corresponds to a new item. Similarly, combining two (items corresponding to) push computations followed by a reduce transition also derives an additional item. Kuhlmann et al. (2011) proved the completeness and unambiguity of the deduction system.

Our deduction system results in an $O(n^4)$ algorithm. In concurrent work Shi et al. (2017) showed that there exists an $O(n^3)$ formulation for transition-systems with the same feature space; by deferring the computation of scores associated with shift transitions to inference rules for reduce transitions, the deduction system has $O(n^2)$ rather than $O(n^3)$ items.

The marginal probability distribution $p(\mathbf{w})$ is computed by defining the inside score $I[l, i, j]$ for every item in the deduction system, which corresponds to the probability of generating $\mathbf{w}_{i+1:j}$ given that $\sigma_1 = l$ (and $\beta = i$). Computing the sentence probability corresponds to deducing the goal, i.e., $p(w_{0:n}) = I[\emptyset, 0, n]$.

The inside algorithm is given in Algorithm 5.[3] The structure of the algorithm is similar to that of the CKY algorithm for constituency parsing. Word probabilities are computed incrementally, so the algorithm can be used for online estimation of next word probabilities

[3]Our implementation uses log probabilities.

given partially observed sentences. The final reduce transition is predicted after the goal configuration $([0], n)$ has been reached.

Algorithm 5 Inside algorithm for the shift-reduce transition-based generative model.

```
1: $I[\emptyset, 0, 1] \leftarrow p_{gen}(w_1|\boldsymbol{h}_0)$
2: for $j = 2, \ldots, n$ do
3:     for $i = 0, \ldots, j-2$ do
4:         $I[i, j-1, j] \leftarrow p_{act}(\mathrm{sh}|\boldsymbol{h}_i, \boldsymbol{h}_{j-1}) p_{gen}(w_j|\boldsymbol{h}_i, \boldsymbol{h}_{j-1})$
5:     end for
6:     for $i = j-2, \ldots, 1$ do
7:         for $k = i+1, \ldots, j-1$ do
8:             $T[k] \leftarrow I[i, k, j] p_{act}(\mathrm{re}|\boldsymbol{h}_k, \boldsymbol{h}_j)$
9:         end for
10:        for $l = 0, \ldots, i-1$ do
11:            $I[l, i, j] \leftarrow \sum_{k=i+1}^{j-1} I[l, i, k] T[k]$
12:        end for
13:    end for
14:    $I[\emptyset, 0, j] \leftarrow \sum_{k=1}^{j-1} I[\emptyset, 0, k] I[0, k, j] p_{act}(\mathrm{re}|\boldsymbol{h}_k, \boldsymbol{h}_j)$
15: end for
16: return $I[\emptyset, 0, n] + p_{act}(\mathrm{re}|\boldsymbol{h}_0, \boldsymbol{h}_n)$
```

To train the model without supervised transition sequences, we optimise the negative log likelihood of $p(w_{0:n})$ directly with gradient-based optimisation. We use automatic differentiation, which is equivalent to computing the gradients with the outside algorithm (Eisner, 2016). To decode with the model, i.e. to find the most likely shift-reduce sequence for a given sentence, we perform Viterbi search over the dynamic program by maximising rather than summing over different split positions (lines 11 and 14 in Algorithm 5) and recover the best transition sequence through back-pointers.

4.3 Transition-based dependency parsing

We now define generative dependency parsing models based on the shift-reduce model proposed above. The shift-reduce system can be interpreted as letting every shift action add a right-arc between the stack top and current index nodes. However, the clear limitation of this interpretation is that no left-arcs can be added.

The arc-eager and arc-hybrid (Kuhlmann et al., 2011) transition systems are based on the same shift-reduce operations, but include additional actions that add left- and right-arcs

Action	State before	State after	Probability	Arc added
Shift	$(S\|i, j)$	$(S\|i\|j, j+1)$	$p_{act}(\text{sh}\|\boldsymbol{h}_i, \boldsymbol{h}_j)p_{gen}(w_{j+1}\|\boldsymbol{h}_i, \boldsymbol{h}_j)$	-
Left-arc	$(S\|l\|i, j)$	$(S\|l, j)$	$p_{act}(\text{la}\|\boldsymbol{h}_i, \boldsymbol{h}_j)$	$j \to i$
Right-arc	$(S\|l\|i, j)$	$(S\|l, j)$	$p_{act}(\text{ra}\|\boldsymbol{h}_i, \boldsymbol{h}_j)$	$l \to i$

Table 4.4: The arc-hybrid transition system.

between words to construct projective dependency trees. Arc-standard differs from these transition systems in that its stack operations are not purely shift-reduce: It can also remove σ_2 while keeping σ_1. Consequently the arc-standard dynamic program has an impractically high time complexity of $O(n^5)$ (before an explicit feature model is added), so we shall not explore it any further here.

4.3.1 Arc-hybrid parser

The arc-hybrid transition system has three actions: Shift, left-arc and right-arc. Left-arc and right-arc are both reduce actions, popping the stack, but they add arcs between different word pairs: Left-arc from β to σ_1 and right-arc from σ_2 to σ_1. The node popped from the stack is always the dependent in the arc; consequently the dependency tree is constructed bottom-up as in arc-standard. The transitions and their probability distributions in the generative model are given in Table 4.4. Left- and right-arc actions also predict arc labels, using another output softmax conditioned on the same context. Note that the dynamic program does not allow us to condition on σ_2, even though it heads the right-arcs that are added. The dynamic program can be extended, but that would further increase computational complexity; our results show that the model works well without it.

For supervised training we optimise the joint probability distribution $p(\mathbf{w}, \mathrm{s})$, where a static oracle is used to derive s from the training examples. Note that there is spurious ambiguity as multiple transition sequences can lead to the same parse tree: Under some circumstances, where a right-arc is a valid transition given the current configuration, it can be delayed by performing shift instead. The right-arc is then added later when the same two words are on top of the stack again. Our oracle adds right-arcs as soon as possible.

The inside algorithm is similar to that of the shift-reduce parser. All decisions are conditioned on the same elements. The reduce probability is obtained by summing the

Action	State before	State after	Probability	Arc added
Shift	$(S\|i^b, j)$	$(S\|i^b\|j^0, j+1)$	$p_{act}(\text{sh}\|\boldsymbol{h}_i, b, \boldsymbol{h}_j)p_{gen}(w_{j+1}\|\boldsymbol{h}_i, b, \boldsymbol{h}_j, \text{sh})$	-
Right-arc	$(S\|i^b, j)$	$(S\|i^b\|j^1, j+1)$	$p_{act}(\text{ra}\|\boldsymbol{h}_i, b, \boldsymbol{h}_j)p_{gen}(w_{j+1}\|\boldsymbol{h}_i, b, \boldsymbol{h}_j, \text{ra})$	$i \to j$
Left-arc	$(S\|l\|i^0, j)$	$(S\|l, j)$	$p_{act}(\text{re}\|\boldsymbol{h}_i, 0, \boldsymbol{h}_j)$	$j \to i$
Reduce	$(S\|l\|i^1, j)$	$(S\|l, j)$	$p_{act}(\text{re}\|\boldsymbol{h}_i, 1, \boldsymbol{h}_j)$	-

Table 4.5: The arc-eager transition system.

left- and right-arc scores. The dynamic program cannot distinguish between left- and right arcs when computing the marginal probability, as the reduce actions result in the same transition system configuration. Therefore while the model can be trained with supervision as an arc-hybrid parser, it cannot be used for unsupervised dependency tree induction.

We perform Viterbi decoding with the dynamic program to find the transition sequence s that maximises $p(\mathbf{s}, \mathbf{w})$. For every reduce decision (line 8 in Algorithm 5) we greedily choose left-arc or right-arc, as the choice has no influence on the rest of the dynamic program. The probability of the highest scoring arc label is included for each reduce prediction. The splitting point for each item reached by a reduce action is recorded, so that the highest scoring derivation can be recovered.

4.3.2 Arc-eager parser

The arc-eager parser was defined in Section 2.3. As noted, the two reduce actions, reduce and left-arc, are always mutually exclusive. To keep track of which actions are valid, here we augment the state configuration to record for each element on the stack whether it is headed or not. A derivation is valid only if all the generated nodes (except the end-of-sentence token) are headed when the terminal configuration is reached, so that the dependency graph is connected. Therefore when a shift action is performed which sets $\beta = n$, all the words on the stack (apart from the root) have to be headed; reduce transitions then follow deterministically. The arc-eager transition actions and their parameterisations are given in Table 4.5.

In order to understand the relationship between the arc-hybrid and arc-eager parsers, we can re-interpret arc-eager's right-arc transition as deciding that an arc should be added, but delaying adding that arc until the dependent is reduced. The dependency tree will then be built in the same way as for arc-hybrid, with the same shift-reduce sequence, the difference being that arc-eager makes the decision to add a right-arc when the dependent is shifted, while arc-hybrid makes it when the dependent is reduced. Also note that a shift prediction in arc-eager is implicitly deciding that a left-arc will have to be attached to the word shifted when it is reduced. The spurious ambiguity in arc-eager is essentially the same as for arc-hybrid.

The dynamic program for the arc-eager parser has to keep track of whether the stack top is headed or not. Items have the form $[l^b, i^c, j]$, where b and c are binary variables indicating whether nodes l and i are headed, respectively. The deduction system, again extending Kuhlmann et al. (2011) is defined as follows:

Goal:

$$[\emptyset, 0^0, n]$$

Axioms:

$$[\emptyset, 0^0, 1]$$

Deduction rules:

$$\begin{aligned} [l^b, i'c, j] &\to [i^c, j^0, j+1] && \text{(shift)} \\ [l^b, i^c, j] &\to [i^c, j^1, j+1] && \text{(right-arc)} \\ [l^b, i^c, k][i^c, k^0, j] &\to [l^b, i^c, j] && \text{(left-arc)} \\ [l^b, i^c, k][i^c, k^1, j] &\to [l^b, i^c, j] && \text{(reduce)} \end{aligned}$$

The inside algorithm for arc-eager parsing is given in Algorithm 6. While Algorithm 5 computes the probabilities incrementally for each sentence position, the inside scores are computed in a different order in Algorithm 6. This enables a more efficient implementation, specifically enabling more parallelization. (see Section 4.4.2). When $\beta = n$, the dynamic program is restricted to allow only reduce transitions, which enforces the property that nodes on the stack should already be headed. The Viterbi algorithm again follows the same structure as the inside algorithm. For every item $[l^b, i^c, j]$ the highest scoring splitting item k^d is recorded, where k is the splitting point. Headedness variable d indicates whether a reduce or left-arc action was performed to reach the item.

Algorithm 6 Inside algorithm for the arc-eager parser.

```
 1: $I[\emptyset, 0^0, 1] \leftarrow p_{gen}(w_1|\boldsymbol{h}_0)$
 2: for $i = 0, \ldots, n-2$ do
 3:     for $j = i+1, \ldots, n-1$ do
 4:         for $c = 0, 1$ do
 5:             $I[i^c, j^0, j{+}1] \leftarrow p_{act}(\mathrm{sh}|\boldsymbol{h}_i, c, \boldsymbol{h}_j)$
 6:                             $\cdot p_{gen}(w_{j+1}|\boldsymbol{h}_i, c, \boldsymbol{h}_j, \mathrm{sh})$
 7:             $I[i^c, j^1, j{+}1] \leftarrow p_{act}(\mathrm{ra}|\boldsymbol{h}_i, c, \boldsymbol{h}_j)$
 8:                             $\cdot p_{gen}(w_{j+1}|\boldsymbol{h}_i, c, \boldsymbol{h}_j, \mathrm{ra})$
 9:         end for
10:     end for
11: end for
12: for $gap = 2, \ldots, n$ do
13:     $I[\emptyset, 0^0, gap] = \sum_{k=1}^{gap-1}(I[\emptyset, 0^0, k]I[0^0, k^1, gap]$
14:                             $\cdot p_{act}(\mathrm{re}|\boldsymbol{h}_k, 1, \boldsymbol{h}_{gap}))$
15:     for $i = 1, \ldots, n - gap$ do
16:         $j = i + gap$
17:         for $c = 0, 1$ do
18:             for $k = i+1, \ldots, j-1$ do
19:                 if $j = n$ then
20:                     $T[k] \leftarrow I[i^c, k^1, j]p_{act}(\mathrm{re}|\boldsymbol{h}_k, 1, \boldsymbol{h}_j)$
21:                 else
22:                     $T[k] \leftarrow (I[i^c, k^0, j]p_{act}(\mathrm{re}|\boldsymbol{h}_k, 0, \boldsymbol{h}_j)$
23:                             $+ I[i^c, k^1, j]p_{act}(\mathrm{re}|\boldsymbol{h}_k, 1, \boldsymbol{h}_j))$
24:                 end if
25:             end for
26:             for $l = 0, \ldots, i-1$ do
27:                 for $b = 0, 1$ do
28:                     $I[l^b, i^c, j] \leftarrow \sum_{k=i+1}^{j-1} I[l^b, i^c, k]T[k]$
29:                 end for
30:             end for
31:         end for
32:     end for
33: end for
34: return $I[\emptyset, 0^0, n] + p_{act}(\mathrm{re}|h_0, h_n)$
```

4.4 Experiments

We aim to evaluate our recurrent generative models trained both with and without parsing supervision. Similar to Chapter 3 we evaluate the models as both parsers and language models. Our goals are to test whether the availability of unbounded contexts and exact decoding leads to more accurate parsing, whether syntactic structure is beneficial for recurrent language models, and whether our models can learn reasonable syntactic structures without supervision.

4.4.1 Setup

We use the same WSJ data and preprocessing as in Section 3.4. Our main supervised experiments are based on SD dependencies, but we also report results on YM.

Our models are implemented in PyTorch,[4] which constructs computation graphs dynamically. During training, sentences are shuffled for each epoch. For unsupervised learning we use mini-batches consisting of sentences of the same length.[5] We base the hyperparameters of our models primarily on the "medium" language model of (Zaremba et al., 2014): A two-layer LSTM with embeddings and hidden states size 650 with dropout of 0.5 on the RNN inputs and outputs. Weights are initialised randomly from the uniform distribution over the range $(-0.05, 0.05)$. Gradient norms are clipped (Pascanu et al., 2013) to 5.0. The SGD learning rate schedules are specified separately for the models below. For training the discriminative baselines we use the Adam optimiser (Kingma and Ba, 2015).

4.4.2 Efficient implementation

We train and execute our models on a GPU. In order to achieve acceptable run-times for our models, especially for unsupervised training, we need to utilise GPUs efficiently, which requires maximising parallelism. We take some guidance from previous work on efficiently implementing CKY parsers on GPUs (Yi et al., 2011; Johnson, 2011; Canny et al., 2013). Our dynamic programs follow a similar structure, but in contrast to constituency parsers

[4] `http://pytorch.org/`

[5] To include all sentences in each epoch, some mini-batches may have less sentences than the given mini-batch size.

Model	Greedy	Exact
Arc-eager (uni)	79.90/77.67	81.37/79.08
Arc-eager (bi)	92.82/90.63	**92.90/90.68**
Arc-hybrid (uni)	84.12/81.54	84.21/81.61
Arc-hybrid (bi)	92.85/90.42	92.89/90.47

Table 4.6: PTB SD development set parsing results with discriminative RNN models, reporting unlabelled and labelled attachment scores (UAS/LAS). RNNs are either unidirectional (uni) or bidirectional (bi).

with large grammars (Petrov et al., 2006), the grammar constant in our approach is negligible.

We pre-compute transition and word emission probabilities before computing the entries in the dynamic programming table. These computations are vectorised over all index pairs. For Viterbi decoding (or running the inside algorithm at test time) the dynamic program is executed on CPU, as we don't have to build a computation graph for backpropagation and all the neural network computations have already been performed. However for unsupervised training the inside algorithm has to be implemented inside the computation graph so that backpropagation can be performed with automatic differentiation. The naive implementation has four nested loops for the $O(n^4)$ algorithm, which creates a bottleneck in constructing the computation graphs and is prohibitively expensive for unsupervised training. Vectorising the two inner loops (lines 16 to 31 in Algorithm 6) results in an order of magnitude speed-up, which enables training in a reasonable time. We also avoid recomputing values by factorising out the computation of $T[k]$ (lines 19-23). Finally, the computation is batched across sentences of the same length, which enables us to maximise the GPU throughput.

For supervised training we also perform batch processing: After the sentences are encoded with an RNN, we extract the inputs to the transition, word and relation prediction models across the batch, and then perform the neural network computations in parallel. The supervised models trains in about 3 minutes per epoch.

Model	UAS/LAS
Kiperwasser and Goldberg (2016)	93.2/91.2
Arc-eager	93.26/91.03
Arc-hybrid	93.29/90.83

Table 4.7: PTB test set parsing results with discriminative bidirectional RNN models.

4.4.3 Supervised parsing

Discriminative parsing We train discriminative baselines (Table 4.6) using the same feature space as the generative models, therefore also enabling exact decoding. Discriminative models are trained with Adam (Kingma and Ba, 2015) with an initial learning rate of 0.0001. Both unidirectional and bidirectional RNNs can be used as encoder. We see that the bidirectional encoder is crucial for accuracy. Exact decoding is only marginally more accurate than greedy decoding (although it is of more benefit for the less accurate unidirectional arc-eager model). This is further evidence of the label bias problem discussed in Section 2.4. Andor et al. (2016) similarly showed that a locally normalised model without lookahead features cannot obtain good performance even with beam-search (81.35% UAS), while their globally normalised model can reach close to optimal performance without look-ahead.

The arc-hybrid model performs better than arc-eager with the unidirectional encoder. A possible explanation for this is that the arc-eager model makes right-arc attachment decisions earlier in the transition sequence than arc-hybrid, which means that it has access to less context to condition those predictions on. When arc-eager decides to shift rather than right-arc, it is also implicitly deciding that the stack top word should become the left dependent of some future word which has not been observed yet.

The performance of our implementation is on par with that of the arc-hybrid transition-based parser of Kiperwasser and Goldberg (2016) (Table 4.7), despite their model using more features, POS tags, externally pre-trained word embeddings, dynamic oracle training and a margin-based loss function. Shi et al. (2017) showed that globally normalised training improves the accuracy of this class of discriminative models.

Model	Greedy	Exact
Arc-eager buffer-next	80.79/78.56	87.34/84.84
Arc-hybrid buffer-next	85.25/82.83	**91.19/88.66**
Arc-Hybrid stack-next	56.98/52.22	82.77/78.01

Table 4.8: PTB SD development set parsing results with supervised generative models, reporting unlabelled and labelled attachment scores (UAS/LAS).

Model	UAS/LAS
HPYP GenDP	87.9/86.2
FFNN GenDP	90.10/87.74
Titov and Henderson (2007)	91.43/89.02
Arc-eager	88.20/85.91
Arc-hybrid	**91.01/88.54**

Table 4.9: PTB SD test set parsing results with supervised generative models. Beam-search results are reported for the models where exact decoding is not possible.

Generative parsing The supervised generative parsers are trained with batch size 16, with an initial SGD learning rate of 1.0, which is divided by 1.7 for every epoch after 6 initial epochs.

Our best recurrent generative model outperforms the Markov generative models proposed in Chapter 3 (Table 4.9). In contrast to the discriminative models, here exact decoding is crucial to performance. The generative models are much more accurate than the unidirectional discriminative models with Viterbi decoding, showing that the word prediction model does benefit parsing accuracy, which would not have been the case if that model only depended on the previous word.

The arc-hybrid model is more accurate than arc-eager, as was the case for the unidirectional discriminative models. Global search and the generative model does not seem to compensate for the arc-eager model's property of conditioning on less context when making right-arc decisions. We also see that the buffer-next model is much more accurate than the stack-next generative formulation, which can be explained by the fact that the stack-next model cannot condition on the buffer word when making transition predictions.

The exact decoding algorithm for this model is also actually faster than the most accurate model with particle filtering decoding: The arc-hybrid model parses 7.4 sentences per

Model	UAS/LAS
HPYP GenDP	88.47/86.13
FFNN GenDP	90.16/88.83
Titov and Henderson (2007)	90.75/89.29
Arc-eager	87.61/86.36
Arc-hybrid	**90.71/88.68**

Table 4.10: PTB YM test set parsing results with supervised generative models.

Model	Perplexity
Interpolated Kneser-Ney 5-gram	170.09
Sequential LSTM (unbatched)	118.69
Sequential LSTM (batched)	100.67
FFNN GenDP	138.62
RNNG (Dyer et al., 2016)	105.2
RNNG (Kuncoro et al., 2017)	101.2
Supervised (SD) shift-reduce buffer-next	111.53
Supervised (SD) shift-reduce stack-next	**107.61**
Unsupervised shift-reduce stack-next	125.20

Table 4.11: Language modelling perplexity results on the PTB parsing test set.

second, against 4 sentences per second for the particle filtering decoder.

For further comparison we also report results on YM dependencies (Table 4.10). For this representation our arc-hybrid model is on par with the best previous generative parser (Titov and Henderson, 2007). The arc-eager model again performs worse.

4.4.4 Language modelling

We apply both the supervised and unsupervised models to language modelling. The supervised models are trained as arc-hybrid parsers; for language modelling arc labels and directionality is not predicted. The unsupervised model is trained with only shift and reduce transitions as latent variable.

We evaluate language models with a sentence i.i.d. assumption, similar to the setup in Chapter 3. In contrast, standard evaluation setups for RNN language models typically treat the entire corpus as single sequence. To evaluate the consequence of the sentence independence assumption, we trained a model on the most widely used PTB language modelling setup (Chelba and Jelinek, 2000; Mikolov et al., 2011), which uses a different training/testing split and preprocessing which limits the vocabulary to 10k. Our baseline LSTM obtains 92.71 test perplexity on this setup, against 78.4 of Zaremba et al. (2014), which uses the same hyperparameters (we use a batch-size of 16) without a sentence i.i.d. assumption.

Results are reported in Table 4.11. As baselines without syntactic structure we use the interpolated Kneser-Ney n-gram model and standard (sequential) LSTMs, trained with or without batching (mini-batch size 16). The initial learning rate is 1.0 for the batched model and 0.1 for the unbatched model.

The LSTM baselines already outperform our best syntactic feed-forward neural model from Chapter 3. We see that there is a significant difference between training with or without mini-batching for the baseline; similarly our model's perplexities also improve when trained with batching. The batched baseline trained performs slightly better than Recurrent Neural Network Grammars (RNNG) (Dyer et al., 2016; Kuncoro et al., 2017), a constituency syntax-based RNN language model trained without batching.[6]

The results show that our syntactic language models do not achieve better performance than the LSTM baseline. We experimented with different dependency representations on the development set, including SD, YM and Universal Dependencies (Nivre et al., 2016) . In contrast to the results of our Markov syntactic models we found little difference in language modelling performance between the dependency representations. The stack-next generative model has lower perplexity than the buffer-next model, which supports the hypothesis that its conditioning contexts are more informative, as they are syntactically more relevant.

Recurrent Neural Network Grammars (RNNG) (Dyer et al., 2016; Kuncoro et al., 2017) uses importance sampling to estimate perplexity, as exact inference is intractable. The advantage of RNNG is that it is able to compute a compositional representation of the stack and the partially constructed parse tree, while our model can only make use of the position on top of the stack and otherwise has to rely on the sequentially computed RNN representations. The disadvantage of RNNG is that inference can only be performed once the entire

[6]Our experimental setup is the same as Dyer et al. (2016), except for a minor implementation difference in unknown word clustering; Dyer et al. (2016) reports 169.31 perplexity on the same IKN model.

Model	U-UAS	P	R	F1
Right-branching	44.72	46.05	22.59	30.31
Shift-reduce	24.25	19.71	25.74	22.33
Arc-eager	28.34	15.43	17.38	16.35

Table 4.12: Unsupervised parsing results: Undirected UAS and unlabelled bracket precision, recall and F1.

sentence has been observed, as the proposal distribution is a discriminative parser. Particle filtering inference, as we proposed in Chapter 3, does not suffer from this limitation. The inference methods of our models in this (and the previous) chapter are able to estimate next word probabilities from partially observed sequences, which RNNG is not able to do.

4.4.5 Unsupervised learning

We perform unsupervised learning with the shift-reduce and arc-eager models. The shift-reduce model learns transition sequences which can be interpreted as generating right-arc only dependency trees. The arc-eager model can learn full dependency trees – however in practice we found that it rarely predicts left-arcs. The models are trained with SGD with batch size 4, initial learning rate 1.0 and a decay factor 2.0, with a mini-batch size of 8. In order to restrict training time we limit the length of training sentences to 40 by cropping (rather than excluding) longer WSJ sentences.

Language modelling results show that the models have worse perplexity than the supervised models and LSTM baselines (Table 4.11). The models are hard to optimise and training time is longer than for the supervised models, which makes finding good hyper-parameters a challenge. The results seem to indicate that the current model does not have a strong enough inductive bias to learn parse trees (or stack transition sequences) which enable it to improve language modelling performance over sequential LSTMs.

We evaluated the parse trees learned by the model against gold standard trees. The accuracy of the trees are worse than a right-branching baseline (where every word is the dependent of the word to its left) on two metrics (Table 4.12). The first metric is undirected unlabelled attachment score (U-UAS). The second is based on the constituent structures corresponding to the dependencies learned: The dependency trees (gold and predicted) are

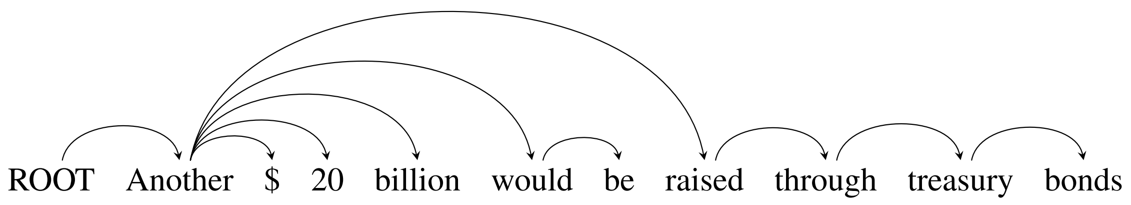


Figure 4.2: Sentence with dependencies induced by the unsupervised model.

converted to unlabelled bracket structures and evaluated with the PARSEVAL metric for constituency trees (Black et al., 1991), following Noji et al. (2016).

The recall of the shift-reduce model is slightly higher than the right-branching baseline, but precision is much lower. Arc-eager performs better than shift-reduce on undirected UAS, but worse on the bracket structure evaluation. The trees predicted by the arc-eager model have a very low percentage of left-arcs, showing that the model is not able to distinguish adequately between left- and right-arcs.

Qualitative analysis shows that overall the model did not succeed in learning informative, non-trivial tree structures – in most cases it learns to attach words either to the immediate previous word or to the root. This seems to indicate that the memorisation ability of the LSTM is too strong to enable the model to learn to rely on the parse structure to make next word predictions. However unsupervised dependency parsers usually require elaborate initialisation schemes or biases to produce non-trivial trees (Klein and Manning, 2004; Spitkovsky et al., 2010a; Bisk and Hockenmaier, 2015). An example dependency tree predicted by the unsupervised model is given in Figure 4.2.

4.5 Conclusion

We proposed a new framework for generative models of syntactic structure based on recurrent neural networks. We presented efficient algorithms for training these models with or without supervision, and to apply them to make online predictions for language modelling through exact marginalisation. Results show that the model obtains state-of-the-art performance among generative models for dependency parsing, although comparative models used much more complex feature spaces. However the models do not obtain better intrinsic language modelling performance than a vanilla RNN on a standard dataset. While the advantage of the models in this chapter is that they enable exact inference, the restrictions that this imposes (conditioning only on the top of the stack, without syntax-based composition) might be too strong to enable useful unsupervised learning.

Chapter 5

Neural Semantic Graph Parsing

Parsing sentences to linguistically-expressive semantic representations is a key goal of Natural Language Processing; yet statistical parsing has focussed predominantly on bilexical dependencies. In this chapter we develop fast, robust and accurate models to parse linguistically deep representations. The main representation we use is Minimal Recursion Semantics (MRS) (Copestake et al., 1995, 2005), which serves as the semantic representation of the English Resource Grammar (ERG) (Flickinger, 2000). Existing parsers for full MRS are grammar-based, performing disambiguation with a maximum entropy model (Toutanova et al., 2005; Zhang et al., 2007); this approach has high precision but incomplete coverage. In the following example, sentence 1 cannot be parsed by the grammar, as it does not account for how the prepositional phrase *from some points of view* should be attached. On the other hand, sentence 2 is acceptable:

1. But this is splendid, really unique from some points of view.

2. But this is a splendid, really unique point of view.

We propose a neural encoder-decoder transition-based parser which is the first full-coverage semantic graph parser for MRS. The model architecture uses stack-based embedding features, predicting graphs jointly with unlexicalised predicates and their token alignments. We do not assume access to the underlying ERG or syntactic structures from which the MRS analyses were originally derived. Instead we develop parsers for two graph-based conversions of MRS, Elementary Dependency Structure (EDS) (Oepen and Lønning, 2006) and Dependency MRS (DMRS) (Copestake, 2009), of which the latter is inter-convertible with MRS.

Abstract Meaning Representation (AMR) (Banarescu et al., 2013) is a graph-based semantic representation that shares the goals of MRS. Aside from differences in the choice of which linguistic phenomena are annotated, MRS is a compositional representation explicitly coupled with the syntactic structure of the sentence, while AMR does not assume compositionality or alignment with the sentence structure. Recently a number of AMR parsers have been developed (Flanigan et al., 2014; Wang et al., 2015a; Artzi et al., 2015; Damonte et al., 2017), but corpora are still under active development and low inter-annotator agreement places an upper bound of 83% F1 on expected parser performance (Banarescu et al., 2013). We apply our model to AMR parsing as well, proposing heuristics to deal with the underspecification of AMR parses relative to MRS-based graphs.

Our parser is based on a transition system for semantic graphs. However, instead of generating arcs over an ordered, fixed set of nodes (the words in the sentence), we generate the nodes and their alignments jointly with the transition actions. One of the main reasons for the prevalence of bilexical dependencies and tree-based representations is that they can be parsed with efficient and well-understood algorithms. However, one of the key advantages of deep learning is the ability to make predictions conditioned on unbounded contexts encoded with RNNs; this enables us to predict more complex structures without increasing algorithmic complexity.

5.1 Semantic Graphs

We define a common framework for semantic graphs in which we place both MRS-based graph representations (DMRS and EDS) and AMR, defined in this section. Sentence meaning is represented with rooted, labelled, directed graphs (Kuhlmann and Oepen, 2016).[1] Examples graphs are shown in Figures 5.1 to 5.3. Node labels are referred to as *predicates* (*concepts* in AMR) and edge labels as *arguments* (*relations* in AMR). In addition *constants*, a special type of node modifiers, are used to denote the string values of named entities and numbers (including date and time expressions). Every node is aligned to a token or a continuous span of tokens in the sentence the graph corresponds to.

[1]We do not assume that the graphs are connected or acyclic.

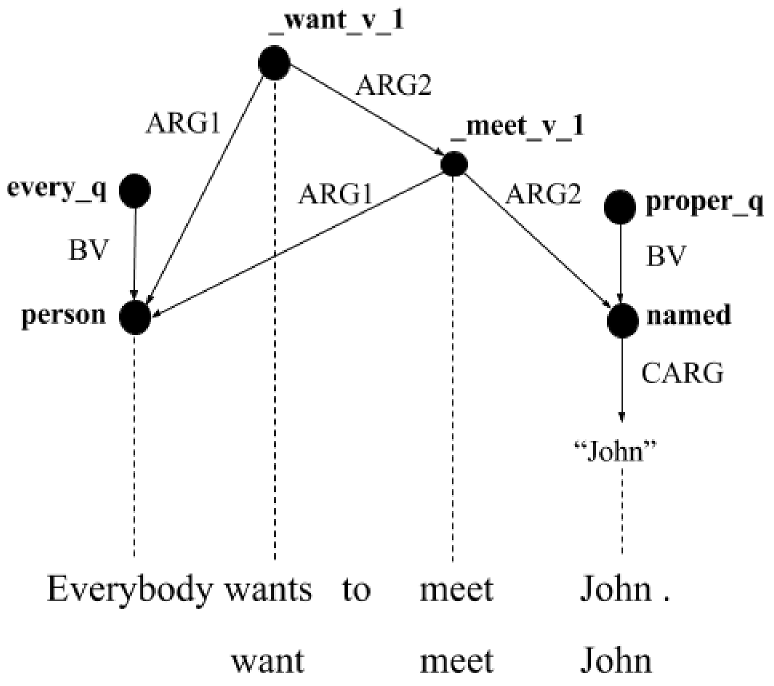


Figure 5.1: Example Elementary Dependency Structure (EDS) graph. The alignments are indicated, alongside the lemmas of surface predicates and constants.

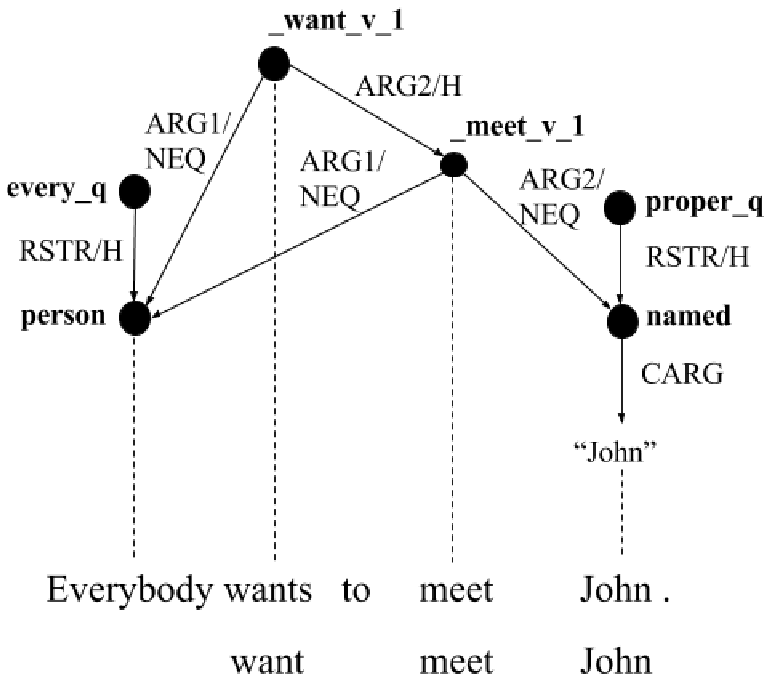


Figure 5.2: Example Dependency Minimal Recursion Semantics (DMRS) graph.

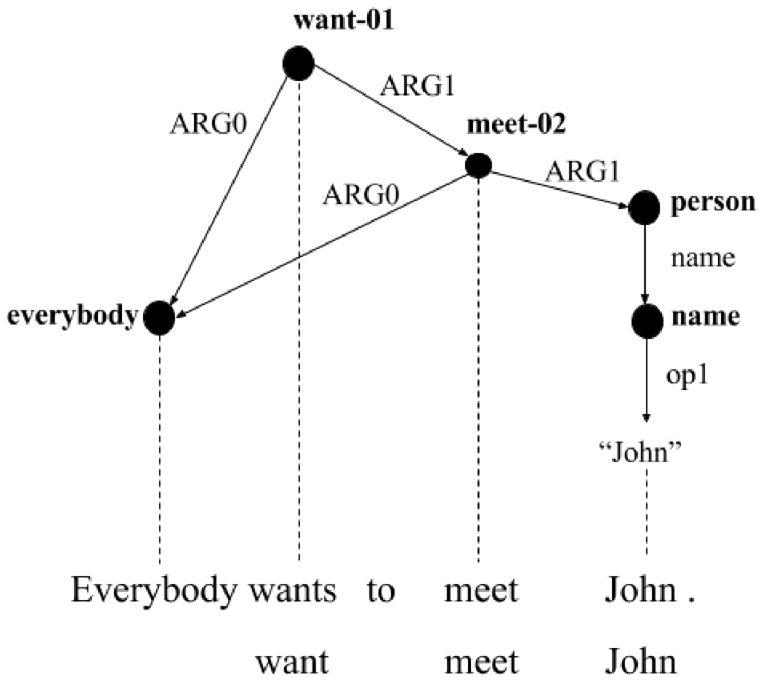


Figure 5.3: Example Abstract Meaning Representation (AMR) graph. Alignments are obtained with an automatic aligner.

5.1.1 Minimal Recursion Semantics

Minimal Recursion Semantics (MRS) is a framework for computational semantics that can be used for parsing or generation (Copestake et al., 2005). Instances and eventualities are represented with logical variables. Predicates take arguments with labels from a small fixed set of roles. Arguments are either logical variables or handles, designated formalism-internal variables. Handle equality constraints support scope underspecification; multiple scope-resolved logical representations can be derived from one MRS structure. A predicate corresponds to its intrinsic argument and is aligned to a character span of the (untokenised) input sentence. Predicates representing named entities or numbers are parameterised by strings. Quantification is expressed through predicates that bound instance variables, rather than through logical operators such as $\exists$ or $\forall$. MRS was designed to be integrated with feature-based grammars such as Head-driven Phrase Structure Grammar (HPSG) (Pollard and Sag, 1994) or Lexical Functional Grammar (LFG) (Kaplan and Bresnan, 1982). MRS has been implemented in the English Resource Grammar (ERG) (Flickinger, 2000), a broad-coverage high-precision HPSG grammar.

Oepen and Lønning (2006) proposed Elementary Dependency Structure (EDS), a conversion of MRS to variable-free dependency graphs which drops scope specification. Copestake (2009) extended this conversion to avoid information loss, primarily through richer edge labels. The resulting representation, Dependency MRS (DMRS), can be converted back to the original MRS or used directly in MRS-based applications (Copestake et al., 2016). We are interested in developing parsers for both of these representations: While EDS is more interpretable as an independent semantic graph representation, DMRS can be related back to underspecified logical forms.[2] A bilexical simplification of EDS has previously been used for semantic dependency parsing (Oepen et al., 2014, 2015). Figure 5.1 illustrates an EDS graph, and Figure 5.2 a DMRS graph.

MRS makes an explicit distinction between surface and abstract predicates (by convention surface predicates are prefixed by an underscore). Surface predicates consist of a lemma followed by a coarse part-of-speech tag and an optional sense label. Predicates absent from the ERG lexicon are represented by their surface forms and POS tags. We convert the character-level predicate spans given by MRS to token-level spans for parsing purposes, but the representation does not require gold tokenisation. Surface predicates usually align with the span of the token(s) they represent, while abstract predicates can span longer segments. In full MRS every predicate is annotated with a set of morphosyntactic features, encoding for example tense, aspect and number information; we do not currently model these features.

Previous parsers Prior work for MRS parsing predominantly predicts structures in the context of grammar-based parsing, where sentences are parsed to HPSG derivations consistent with the grammar, in this case the ERG (Flickinger, 2000). The nodes in the derivation trees are feature structures, from which MRS is extracted through unification. This approach fails to parse sentences for which no valid derivation is found. Even though the ERG is a large, broad-coverage grammar, coverage is around 85% on newspaper text, and lower on less formal types of text. Maximum entropy models are used to score the derivations in order to find the most likely parse (Toutanova et al., 2005). This approach is implemented in the PET (Callmeier, 2000) and ACE[3] parsers.

[2]DMRS graphs may have undirected edges.

[3]`http://sweaglesw.org/linguistics/ace/`

There have been some efforts to develop robust MRS parsers. One proposed approach learns a PCFG grammar to approximate the HPSG derivations (Zhang and Krieger, 2011; Zhang et al., 2014). MRS is then extracted with robust unification to compose potentially incompatible feature structures, although that still fails for a small proportion of sentences. The model is trained on a large corpus of Wikipedia text parsed with the grammar-based parser. Ytrestøl (2012) proposed a transition-based approach to HPSG parsing that produces derivations from which both syntactic and semantic (MRS) parses can be extracted. The parser has an option not to be restricted by the ERG. However, neither of these approaches have results available that can be compared directly to our setup, or generally available implementations.

5.1.2 Abstract Meaning Representation

Abstract Meaning Representation (AMR) (Banarescu et al., 2013) graphs can be represented in the same framework, despite a number of linguistic differences with MRS. Some information annotated explicitly in MRS is latent in AMR, including alignments and the distinction between surface (lexical) and abstract concepts.

AMR predicates are based on PropBank (Palmer et al., 2005), annotated as lemmas plus sense labels, but they form only a subset of concepts. Other concepts are either English words or special keywords, corresponding to overt lexemes in some cases but not others. An example AMR graph is shown in Figure 5.3.

AMR makes no assumptions about the relation between an AMR and the structure of the sentence it represents: The representation is not assumed to have any relation to the sentence syntax, no alignments are given and no distinction is made between concepts that correspond directly to lexemes in the input sentences and those that don't. This underspecification creates significant challenges for training an end-to-end AMR parser, which are exacerbated by the relatively small sizes of available training sets. Consequently most AMR parsers are pipelines that make extensive use of additional resources.

Previous parsers Most AMR parsers make assumptions particular to AMR and rely on extensive external (AMR-specific) resources. Therefore they cannot be applied directly to more general structures such as MRS-based graphs.

Flanigan et al. (2014) proposed a two-stage parser that first predicts concepts or subgraphs corresponding to sentence segments, and then parses these concepts into a graph structure. However MRS has a large proportion of abstract nodes which cannot be predicted from short segments and interact closely with the graph structure. Wang et al. (2015a,b) proposed a custom transition system for AMR parsing that converts dependency trees to AMR graphs, relying on assumptions on the relationship between these. Pust et al. (2015) proposed a parser based on syntax-based machine translation (MT), while AMR has also been integrated into CCG Semantic Parsing (Artzi et al., 2015; Misra and Artzi, 2016).

Neural encoder-decoders have been proposed for AMR parsing, but reported accuracies are well below the state-of-the-art (Barzdins and Gosko, 2016), even with sophisticated preprocessing and categorization (Peng et al., 2017b). Foland Jr and Martin (2016) proposed a pipeline consisting of multiple LSTMs, while Damonte et al. (2017) introduced a transition-based parser based on neural network classifiers inside a feature- and resource-rich parser. These parsers perform competitively with state-of-the-art approaches.

5.1.3 Other representations

There are a number of other linguistically-expressive representations with a semantic component and at least partly overlapping goals with the representations we work with in this chapter. The Combinatory Categorial Grammar (CCG) (Steedman, 2000) annotation of the PTB (Hockenmaier and Steedman, 2007) has a semantic bilexical dependency representation, while a line of research uses CCG to assign logical expressions in the context of semantic parsing of questions for query execution (Zettlemoyer and Collins, 2005; Kwiatkowski et al., 2010; Artzi and Zettlemoyer, 2013). The Prague Dependency Treebank (Böhmová et al., 2003) has a tectogrammatical layer which is a rich semantic representation. In the context of HPSG the Enju grammar (Miyao and Tsujii, 2008) is based on an automatic conversion of PTB constituency trees, and bilexical predicate-argument structures can be derived from the annotations. While the Enju parser has very high coverage (97% of PTB sentences (Matsuzaki et al., 2007)), the semantic structures produced are considerably less informative than those given by MRS in the ERG. Universal Conceptual Cognitive Annotation (UCCA) (Abend and Rappoport, 2013) is another semantic representation which, similar to AMR, is not coupled with syntactic representation, and has the

```
:root( <2> _v_1
  :ARG1( <1> person
    :BV-of( <1> every_q ) )
  :ARG2 <4> _v_1
    :ARG1*( <1> person
    :ARG2( <5> named_CARG
      :BV-of ( <5> proper_q ) ) )
```

Figure 5.4: A depth-first traversal of the EDS graph in Figure 5.1 (using unlexicalised predicates) that can be predicted incrementally.

goal to be cross-lingually applicable. See Abend and Rappoport (2017) for a recent review of semantic representations.

5.2 Incremental prediction of graph traversals

The first approach we consider to make graph parsing amenable to incremental prediction is to linearise graphs as the pre-order traversal of their spanning trees, starting at a designated root node (see Figure 5.4). Vinyals et al. (2015b) showed that constituency parsing can be performed with neural sequence-to-sequence models by predicting nested bracketing structures as output sequences. This approach was also applied to logical form prediction (Dong and Lapata, 2016; Jia and Liang, 2016).

While these approaches deal with tree structures, nodes in semantic graphs may have multiple heads. In AMR datasets, graphs are represented as spanning trees with designated root nodes. Edges whose direction in the spanning tree is reversed are marked by adding "-of" to the argument label. Edges not included in the spanning tree (called *reentrancies*) are indicated by adding dummy nodes pointing back to the original nodes. Recent work has also approach AMR parsing as a sequence prediction task (Barzdins and Gosko, 2016; Peng et al., 2017b; Konstas et al., 2017), using variants of this representation.

Our sequential representation (linearisation) is similar, although slightly different tokenisation choices are made. Nodes are identified through their concepts rather than explicit node identifiers, as in AMR datasets. Constants are also treated as nodes. Reentrancy edges are marked with $\star$. Our potentially lossy representation represents these edges by repeating the dependent node labels and alignments. During post-processing reentrancies

```
:focus( want-01
  :ARG0( everybody )
  :ARG1( meet-02
    :ARG0*( everybody )
    :ARG1( person
      :name( name
        :op1( "John" ) ) ) ) )
```

Figure 5.5: *Standard* linearised representation of the AMR in Figure 5.3.

are recovered heuristically by finding the closest nodes in the sequential representation with the same concepts. The alignment does not influence the node ordering.

We propose two approaches for predicting surface predicates. In the first representation (*lexicalised*) the full predicate is always predicted. The second (*delexicalised*) factorises lemmas out of surface predicates: Candidate lemmas are predicted separately for input tokens; surface predicates are represented only by their sense labels. The predicted alignments are used to recover the full surface predicates.

For MRS we extract a dictionary mapping words to lemmas from the ERG lexicon. Candidate lemmas are predicted using this dictionary or, where no dictionary entry is available, with a lemmatiser. The same approach is applied to predict constants, along with additional normalisations such as mapping numbers to digit strings.

Multi-word expressions are handled as follows: Where the lemma of a predicate is a multi-word expression we assume that the first word in the expression will correspond to the lemma of the token corresponding to the start span of the token.[4] Our delexicalisation then lets the remaining words in the expression be part of the predicate predicted by the parser (along with the sense label). This approach allows us to deal with ambiguity in recognizing whether a substring is a multi-word expression or not.

AMR graphs do not distinguish between surface and abstract concepts, and does not include alignments. Our incremental representation that does not add any additional information is referred to as *standard*. An example is given in Figure 5.5.

We propose lexicalised and delexicalised representations for AMR by heuristically classifying concepts as surface or abstract, similar to MRS predicates. We assume that each

[4]This fails for a very small proportion of predicates, the most notable example being the predicate with the lemma “in order to” which corresponds to the token “to.”

```
:focus( <2> -01
  :ARG0( <1> -u )
  :ARG1( <4> -02
    :ARG1*( <1> -u )
    :ARG2( <5> person
      :name( <5> name
        :op1( <5> CONST ) ) ) ) )
```

Figure 5.6: *Delexicalised* linearisation, with alignments, of the AMR in Figure 5.3.

word in a sentence aligns to at most one lexical node in its AMR graph. Where multiple nodes are aligned to the same token, usually forming a subgraph, the lowest element is taken to be the lexical concept. Surface concepts are typically English words in lemma form, which can be predicted with high accuracy from the words they align to. In our linearisations concepts are annotated as being either surface or abstract.

Every constant is replaced with a placeholder `CONST` token; the constant string is recovered as a post-processing step through the predicted token alignment. The delexicalised representation factorises lemmas out of the linearisation so that they are represented by their alignments and sense labels, e.g. `-01` for predicates and `-u` for other concepts. See Figure 5.6 for an example.

5.3 Transition-based parsing

Semantic graphs (e.g. Figure 5.1) are dependency graphs whose nodes are predicates aligned to sentence tokens. Transition-based parsing (Nivre, 2008) has been used extensively to predict dependency graphs incrementally. We apply a variant of the arc-eager transition system that has been proposed for graph (as opposed to tree) parsing (Sagae and Tsujii, 2008; Titov et al., 2009; Gómez-Rodríguez and Nivre, 2010) to derive a transition-based parser for semantic graphs. Here we need to generate the nodes incrementally as the transition system proceeds, conditioning the prediction on the given sentence. Damonte et al. (2017) proposed an arc-eager AMR parser, but their transition system is more narrowly restricted to AMR graphs.

Our transition system is based on the arc-eager graph parser (Section 2.3.2). Here we extended the system defined for planar graphs to arbitrary non-planar graphs. To predict

Action	Stack	Buffer	Arc added
init(1, person)	[ ]	(1, 1, person)	-
sh(1, every_q)	[(1, 1, person)]	(2, 1, every_q)	-
la(BV)	[(1, 1, person)]	(2, 1, every_q)	(2, BV, 1)
sh(2, _v_1)	[(1, 1, person), (2, 1, every_q)]	(3, 2, _v_1)	-
re	[(1, 1, person)]	(3, 2, _v_1)	-
la(ARG1)	[(1, 1, person)]	(3, 2, _v_1)	(3, ARG1, 1)
sh(4, _v_1)	[(1, 1, person), (3, 2, _v_1)]	(4, 4, _v_1)	-
ra(ARG2)	[(1, 1, person), (3, 2, _v_1)]	(4, 4, _v_1)	(3, ARG2, 4)
re	[(1, 1, person)]	(4, 4, _v_1)	-
la(ARG1)	[(1, 1, person)]	(4, 4, _v_1)	(4, ARG1, 1)
re	[ ]	(4, 4, _v_1)	-
sh(5, named_CARG)	[(4, 4, _v_1)]	(5, 5, named_CARG)	-
ra(ARG2)	[(4, 4, _v_1)]	(5, 5, named_CARG)	(4, ARG2, 5)
re	[ ]	(5, 5, named_CARG)	-
sh(5, proper_q)	[(5, 5, named_CARG)]	(6, 5, proper_q)	-
sh(5, proper_q)	[(5, 5, named_CARG)]	(6, 5, proper_q)	-
la(BV)	[(5, 5, named_CARG)]	(6, 5, proper_q)	(6, BV, 5)
re	[ ]	(6, 5, proper_q)	-
sh(EOS)	[(6, 5, proper_q)]	-	-

Figure 5.7: Transition sequence for parsing the graph in Figure 5.1. The transitions are shift (`sh`), reduce (`re`), left arc (`la`) and right arc (`ra`). The action taken at each step is given, along with the state of the stack and buffer after the action is applied, and any arcs added. Shift transitions generate predicates placed on the buffer and their alignments to words in the sentence. Items on the stack and buffer have the form (*node index, alignment, predicate label*), and arcs are of the form (*head index, argument label, dependent index*).

crossing arcs we add a transition, called *cross-arc*, which adds an arc between the buffer and an arbitrary node on the stack (excluding the stack top and root node), whose stack position is also predicted. Additionally we define a special root transition that adds the root arc to the top of the buffer (to be executed just before the node is shifted). This is motivated by the observation that arcs from the root node to the focus (semantic head) of the sentence are often crossing. For DMRS parsing a transition to add undirected arcs is defined (this is possible due to the insensitivity of the rest of the transition system to directionality).

The transition system jointly predicts the predicate labels and alignments of nodes that are generated by shift transitions. *Shift* predicts the predicate and the start of the aligned span of the next node on the buffer, and *reduce* predicts the span end of the popped node. A node's span end alignment often covers the phrase that it heads (e.g. for quantifiers); this gives a natural interpretation to the reduce action.

To derive an oracle for this transition system, it is first necessary to determine the order in which the nodes are generated. We consider two approaches. The first ordering is obtained by performing an in-order traversal of the spanning tree, where the node order is determined by the alignment. In the resulting linearisation the only potentially non-planar arcs are reentrancies. The second approach results in a monotone (non-decreasing) order with respect to the alignments, while following the in-order ordering for nodes with the same alignment.

After determining the node order, we use the arc-eager oracle for planar graphs (Gómez-Rodríguez and Nivre, 2010). The oracle is extended to predict crossing edges using the *cross-arc* transition as follows: Suppose that a node a is on top of the stack and all dependencies (either left or right) between the a and the nodes that follow it in the graph traversal have been added. It is not possible for the planar transition actions to add arcs between the node and other nodes on the stack. If there are dependencies between that node and nodes to its left which have not yet been added, then those nodes have to be on the stack (as the oracle only reduce nodes once all their dependents have been added); therefore in that case the oracle predicts *cross-arc* to add those arcs. Figure 5.7 shows an example transition sequence together with the stack and buffer after each step.

5.4 Encoder-decoder models

We parse sentences to their meaning representations by incrementally predicting semantic graphs together with their alignments. Let $\mathbf{w} = w_1, w_2, \ldots, w_n$ be a tokenised English sentence, $\mathbf{s} = s_1, s_2, \ldots, s_m$ a sequential representation of its graph derivation and $\mathbf{a} = a_1, a_2, \ldots, a_m$ an alignment sequence consisting of integers in the range $1, \ldots, n$. This approach is applicable to both the top-down graph linearisation and transition-based parsing, although we propose one architecture which is designed specifically for transition-based parsing by modelling its parsing configuration.

We model the conditional distribution

$$p(\mathbf{s}, \mathbf{a}|\mathbf{w}) = \prod_{j=1}^{m} p(a_j|(\mathbf{a}, \mathbf{s})_{1:j-1}, \mathbf{w}) p(s_j|\mathbf{a}_{1:j}, \mathbf{s}_{1:j-1}, \mathbf{w}). \tag{5.1}$$

The end-of-span alignments $\mathbf{a}^{(\mathrm{e})}$ are predicted following every reduce action or close bracket. The notion of modelling alignments in sequence transduction models can be traced back to the seminal work of Brown et al. (1993) on statistical machine translation.

5.4.1 Sentence encoder

The sentence $\mathbf{w}$ is encoded with a bidirectional RNN. We use the standard LSTM architecture. For every token w we embed its word, POS tag and named entity (NE) tag as vectors $\boldsymbol{x}_w$, $\boldsymbol{x}_t$ and $\boldsymbol{x}_n$, respectively.

The embeddings are concatenated and passed through a linear transformation

$$g(w) = \boldsymbol{W}^{(e)}[\boldsymbol{x}_w; \boldsymbol{x}_t; \boldsymbol{x}_n] + \boldsymbol{b}^{(e)}, \tag{5.2}$$

such that $g(w)$ has the same dimension as the LSTM cell. Every input position i is represented by a hidden state $\boldsymbol{h}_i$, which is the concatenation of its forward and backward LSTM state vectors.[5]

5.4.2 Hard attention decoder

For the arc-eager model, a_j corresponds to the alignment of the node of the buffer after action s_j is executed. The distribution of s_j is over all transitions and predicates (corresponding to shift transitions), predicted with a single softmax.

[5]Here $\boldsymbol{h}$ includes both the hidden and memory vector of the LSTM state.

The parser output is predicted by an RNN decoder. Let $\boldsymbol{d}_j$ be the decoder hidden state at output position j. We initialise $\boldsymbol{d}_0$ with the final state of the backward encoder.

Alignments are predicted with a pointer network (Vinyals et al., 2015a). The pointer network is a sequence-to-sequence RNN architecture in which the output sequence consistent of indexes (pointers) to the input sequence. The pointer logits are computed with an MLP scoring the decoder hidden state against each of the encoder hidden states (for $i = 1, \ldots, n$),

$$u_j^i = \boldsymbol{w}^T \tanh(\boldsymbol{W}^{(ah)} \boldsymbol{h}_i + \boldsymbol{W}^{(ad)} \boldsymbol{d}_j). \tag{5.3}$$

The alignment distribution is then estimated as

$$p(a_j = i | \mathbf{a}_{1:j-1}, \mathbf{s}_{1:j-1}, \mathbf{w}) = \text{softmax}_i(\boldsymbol{u}_j). \tag{5.4}$$

To predict the next transition s_j, the output vector $\boldsymbol{o}_j$ is computed conditioned on the encoder state vector h_{a_j}, corresponding to the alignment:

$$\boldsymbol{o}_j = \boldsymbol{W}^{(os)} \boldsymbol{d}_j + \boldsymbol{W}^{(oh)} \boldsymbol{h}_{a_j} \tag{5.5}$$

$$\boldsymbol{v}_j = \boldsymbol{R}^{(d)} \boldsymbol{o}_j + \boldsymbol{b}^{(d)}, \tag{5.6}$$

where $\boldsymbol{R}^{(d)}$ and $\boldsymbol{b}^{(d)}$ are the output representation matrix and bias vector, respectively.

The transition distribution is then given by

$$p(s_j | \mathbf{a}_{1:j}, \mathbf{s}_{1:j-1}, \mathbf{w}) = \text{softmax}_{s_j}(\boldsymbol{v}_j). \tag{5.7}$$

Let $e(s_j)$ be the embedding of decoder symbol s_j. The RNN state at the next time step is computed as

$$\boldsymbol{r}_{j+1} = \boldsymbol{W}^{(de)} e(s_j) + \boldsymbol{W}^{(dh)} \boldsymbol{h}_{a_j} \tag{5.8}$$

$$\boldsymbol{d}_{j+1} = \text{RNN}(\boldsymbol{r}_{j+1}, \boldsymbol{d}_j). \tag{5.9}$$

The end-of-span alignment $a_j^{(e)}$ for MRS-based graphs is predicted with another pointer network. The end alignment of a token is predicted only when a node is reduced from the stack, therefore this alignment is not observed at each time step; it is also not fed back into the model.

Our alignment model can be seen as a neural version of IBM model 2 (Brown et al., 1993), in which the alignment distribution depends only on the output symbol position and sentence length. In our neural model the alignment distribution depends on the RNN decoder state as well as all the input words, while the output symbol distribution depends on the RNN representation of the aligned word as well as the decoder RNN state. In contrast to machine translation, we have supervised alignments available to train on.

The hard attention approach can be contrasted to soft attention, which learns to attend over the input without supervision. The attention is computed similarly as

$$\alpha_j^i = \mathrm{softmax}_i(\boldsymbol{u}_j). \tag{5.10}$$

However instead of making a hard selection, a weighted average over the encoder vectors is computed as

$$\boldsymbol{q}_j = \sum_{i=1}^{i=n} \alpha_j^i \boldsymbol{h}_i. \tag{5.11}$$

This vector is used for prediction and fed to the next time step, instead of $\boldsymbol{h}_{a_j}$.

Another alternative is to combine hard and soft attention by using both $\boldsymbol{q}_j$ and $\boldsymbol{h}_{a_j}$ for prediction and feeding either $\boldsymbol{q}_j$ or both into $\boldsymbol{d}_{j+1}$.

5.4.3 Stack-based architecture

We extend the hard attention model to include features based on the transition system stack. These features are embeddings from the bidirectional RNN encoder, corresponding to the alignments of the nodes on the buffer and on top of the stack. This approach is similar to the features proposed by Kiperwasser and Goldberg (2016) and Cross and Huang (2016) for dependency parsing, although they do not use RNN decoders.

In order to implement these features the layer that computes the output vector is extended to

$$\boldsymbol{o}_j = \boldsymbol{W}^{(od)}\boldsymbol{d}_j + \boldsymbol{W}^{(oh)}\boldsymbol{h}_{a_j} + \boldsymbol{W}^{(ot)}\boldsymbol{h}_{\mathrm{st}_0}, \tag{5.12}$$

where $\mathtt{st}_0$ is the sentence alignment index of the element on top of the stack. The input layer to the next RNN time step is similarly extended to

$$\boldsymbol{r}_{j+1} = \boldsymbol{W}^{(de)}e(s_j) + \boldsymbol{W}^{(dh)}\boldsymbol{h}_b + \boldsymbol{W}^{(ds)}\boldsymbol{h}_{\mathrm{st}_0}, \tag{5.13}$$

where b is the buffer alignment after s_j is executed.

Our implementation of the stack-based model enables batch processing in static computation graphs, similar to Bowman et al. (2016). We maintain a stack of alignment indexes for each element in the batch, as well as the buffer alignment indexes. These data structures are updated inside the computation graph after each parsing action. The challenge lies in representing these structures as fixed-sized vectors or tensors and implementing (batched) operations on them. The advantage of this approach is that it enables mini-batch SGD training as well as efficient batch decoding.

5.4.4 Decoding

We perform greedy decoding. For the stack-based architecture we ensure that if the stack is empty, the next transition has to be a shift. For the other models we ensure that the output is well-formed during post-processing by robustly skipping over out-of-place symbols or inserting missing ones. For the top-down AMR models, repeated occurrences of sibling subtrees are removed when equivalent up to the argument number of relations.

5.5 Graph banks and preprocessing

5.5.1 MRS

The MRS dataset we use for our experiments is DeepBank (Flickinger et al., 2012), an HPSG and MRS annotation of the Wall Street Journal (WSJ) corpus. We use version 1.1, corresponding to ERG `1214`[6], and follow the suggested split of sections 0 to 19 as training data, 20 for development and 21 for testing. The gold-annotated training data consists of 35,315 sentences. We use the LOGON environment[7] and the pyDelphin library[8] to extract DMRS and EDS graphs.

DeepBank was developed following an approach known as dynamic treebanking (Oepen et al., 2004) that couples treebank annotation with grammar development, in this case of the ERG. This approach has been shown to lead to high inter-annotator agreement: 0.94 against 0.71 for AMR (Bender et al., 2015). The disadvantage is that parses are only provided for sentences for which the ERG has an analysis, and that the analysis is acceptable

[6]`http://svn.delph-in.net/erg/tags/1214/`

[7]`http://moin.delph-in.net/LogonTop`

[8]`https://github.com/delph-in/pydelphin`

to the annotator. This means that we cannot evaluate parsing accuracy for sentences which the ERG cannot parse (approximately 15% of the original corpus).

We use the Stanford CoreNLP toolkit (Manning et al., 2014) to tokenise and lemmatise sentences, and to perform Parts-of-Speech tagging (Toutanova et al., 2003) and Named Entity Recognition (Finkel et al., 2005). The tokenisation is customised to correspond closely to the ERG tokenisation; hyphens are removed during preprocessing, as they tend to lead to violations of our principle that a token cannot be aligned to multiple surface predicates. This approach leads to a minimal loss of oracle accuracy as token spans have to be converted back to character spans for evaluation.

We use the ERG lexicon (ERG version `1214`) to lemmatise words when they occur in the lexicon, and the Stanford lemmatiser for remaining words. The lemmas are only used to predict the lemma part of predicates in delexicalised models; surface forms are still given as input to the parser.

The ERG lexicon is also used as a dictionary to map tokens (including multi-word expressions) to constant strings where they occur in the lexicon. The lexicon is used as constants are sparse and we do not expect all forms to occur in the training data. We inject knowledge of whether tokens correspond to lexical entries or not in our model as follows: The POS tags given as input to the model are annotated with whether the token has an entry in the predicate lexicon or not; similarly NE tags are annotated with whether a substring starting at that token maps to a constant entry in the lexicon. This enables the model to make better predictions for words which do occur in the training data but do appear in the ERG lexicon.

5.5.2 AMR

We use a number of datasets for AMR parsing. The first is `LDC2015E86`, the dataset released for the SemEval 2016 AMR parsing Shared Task (May, 2016). The main sources of this data are discussion forum and newswire ("proxy") text. The training set has 16,144 sentences. We obtain alignments using the rule-based JAMR aligner (Flanigan et al., 2014). In order to evaluate the domain-adaptability of our approach we also experiment with the Bio AMR Corpus, a smaller dataset in the biomedical domain with 5,542 training sentences.

Candidate Type	JAMR alignments	PropBank	WordNet	NE Tagger	Lemmatiser
Predicates	✓	✓	✓	✗	✓
Other concepts	✓	✗	✓	✗	✓
Constants	✓	✗	✗	✓	✓
Wikification	✓	✗	✗	✓	✗

Table 5.1: Lexical resources used to predict candidate lemmas for different types of AMR outputs. The left-most resource that has a prediction available is used.

We also use CoreNLP to preprocess sentences for AMR parsing.[9] The training data is aligned with the rule-based JAMR aligner (Flanigan et al., 2014). However, our approach requires all nodes to be aligned to a single token, which JAMR is not guaranteed to give. We align every Wiki node to the token with the highest prefix overlap. Other nodes without alignments are aligned to the left-most alignment of their children (if they have any), otherwise to that of their parents. JAMR aligns multi-word named entities as single subgraph to token-span alignments. We split these alignments to be one-to-one between tokens and constants. For other nodes with multi-token alignments we use the start of the given span.

For every token we predict candidate lexemes using a number of lexical resources. A summary of the resources used for each lexical type is given in Table 5.1. The first resource is dictionaries extracted from the aligned training data of each type, mapping each token or span of tokens to its most likely concept lemma or constant. A similar dictionary is extracted from PropBank (Palmer et al., 2005) frames (as included in `LDC2016E25`) for predicate lemmas. Next we use WordNet (Miller, 1995), as available through NLTK (Bird et al., 2009), to map words to verbalised forms (for predicates) or nominalised forms (for other concepts) via their synsets, where available. To predict constant strings corresponding to unseen named entities we use the forms predicted by the Stanford NE tagger, which are broadly consistent with the conventions used for AMR annotation. The same procedure converts numbers to numerals. We use SUTime (Chang and Manning, 2012) to extract normalised forms of dates and time expressions. These preprocessing steps are particularly important due to the small scale of the AMR datasets; a large proportion of concepts and constants in the test data do not occur in the training data and our goal is to mimic the

[9]The Stanford tokenisation (with our customisations) corresponds more closely to AMR concepts and constants than other tokenisers we experimented with, especially due to its handling of hyphenation in the biomedical domain.

surface form transformations that are typically performed to predict AMR outputs.

Input sentences and output graphs in the training data are preprocessed independently. This introduces some noise into the training data, but makes the setup comparable to the way that decoding is performed. The (development set) oracle graph overlap accuracy is 98.7% F1 for the standard representation, 96.16% for the aligned lexicalised representation and 93.48% for the unlexicalised representation.

5.6 Experiments

5.6.1 Model setup

Our parser[10] is implemented in TensorFlow (Abadi et al., 2015). For training we use Adam (Kingma and Ba, 2015) with learning rate 0.01 and batch-size 64. Gradient norms are clipped to 5.0 (Pascanu et al., 2013). We use single-layer LSTMs with dropout of 0.3 (tuned on the development set) on input and output connections. We use encoder and decoder embeddings of size 256, and POS and NE tag embeddings of size 32. For DMRS and EDS graphs the hidden state size is set to 256, while for AMR it is 128 for the small training sets (`LDC2015E86` and Bio AMR) and 256 for the larger or combined datasets. This configuration was found using a combination of grid search and heuristic search within the range of models that fit into a single GPU, and gave the best performance on the development set under multiple graph linearisations. Encoder word embeddings are initialised (in the first 100 dimensions) with pre-trained order-sensitive embeddings (Ling et al., 2015). Singletons in the encoder input are replaced with an unknown word symbol with probability 0.5 for each iteration.

5.6.2 Evaluation

Dridan and Oepen (2011) proposed an evaluation metric called Elementary Dependency Matching (EDM) for MRS-based graphs. EDM computes the F1-score of tuples of predicates and arguments, which are extracted from the gold and predicted graphs. A predicate tuple consists of the label and character span of a predicate, while an argument tuple consists of the character spans of the head and dependent nodes of the relation, together with

[10]Code and data preparation scripts are available at `https://github.com/janmbuys/DeepDeepParser`.

Model	EDM	EDM_P	EDM_A
Top-down lexicalised	81.44	85.20	76.87
Top-down delexicalised	81.72	85.59	77.04
Arc-eager lexicalised	81.35	85.79	76.02
Arc-eager delexicalised	82.56	86.76	77.54

Table 5.2: DMRS development set results for attention-based encoder-decoder models with alignments encoded in the linearisation, for top-down and arc-eager linearisations, and lexicalised and delexicalised predicate prediction.

the argument label. In order to tolerate subtle tokenisation differences with respect to punctuation, we allow span pairs whose ends differ by one character to be matched.

The Smatch metric (Cai and Knight, 2013), proposed for evaluating AMR graphs, also computes tuple-based graph over F1-scores, but does not rely on sentence alignments to determine correspondences between graph nodes. Instead inference is performed over alignments between nodes in the predicted and graph graphs to estimate the maximum F1-score obtainable from a one-to-one node matching. As the matching problem is NP-complete, Cai and Knight (2013) proposed a hill-climbing method, shown to be efficient and accurate when using smart initialisation and random restarts.

5.6.3 MRS parsing

We compare different linearisations and model architectures for parsing DMRS on the development data, showing that our approach is more accurate than baseline neural approaches. We report EDM scores, including scores for predicate (EDM_P) and argument (EDM_A) prediction.

First we report results using standard attention-based encoder-decoders, with the alignments encoded as token strings in the linearisation. (Table 5.2). We compare top-down and arc-eager linearisations, as well as the effect of delexicalising the predicates (factorising lemmas out of the linearisation and predicting them separately.) In both cases constants are predicted with a dictionary lookup based on the predicted spans. A special label is predicted for predicates not in the ERG lexicon – the words and POS tags that make up those predicates are recovered through the alignments during post-processing.

The arc-eager delexicalised representation gives the best performance, even though the

Model	EDM	EDM$_P$	EDM$_A$
Top-down soft attention	81.53	85.32	76.94
Top-down hard attention	82.75	86.37	78.37
Arc-eager hard attention	84.65	87.77	80.85
Arc-eager stack-based	85.28	88.38	81.51

Table 5.3: DMRS development set results of encoder-decoder models with pointer-based alignment prediction, delexicalised predicates and hard or soft attention.

model has to learn to model the transition system stack through the recurrent hidden states without any supervision of the transition semantics. The delexicalised models are more accurate, mostly due to their ability to generalise to sparse or unseen predicates occurring in the lexicon. For the arc-eager representation, the oracle EDM is 99% for the lexicalised representation and 98.06% for the delexicalised representation. The remaining errors are mostly due to discrepancies between the tokenisation used by our system and the ERG tokenisation. The delexicalised models are also faster to train, as the decoder's output vocabulary is much smaller, reducing the expense of computing softmax functions over large vocabularies and requiring less epochs to converge.

Next we consider models with delexicalised linearisations that predict the alignments with pointer networks, contrasting soft and hard attention models (Table 5.3). The results show that the arc-eager models are more accurate than those based on top-down representations. For the arc-eager models we use hard attention, due to the natural interpretation of the alignment prediction corresponding to the transition system. The stack-based architecture gives further improvements.

When comparing the effect of different predicate orderings on the arc-eager model, we find that the monotone ordering performs 0.44 EDM better than the in-order ordering, despite having to parse more non-planar dependencies.

We also trained models that only predict predicates (in monotone order) together with their start spans. The hard attention model obtains 91.36% F1 on predicates together with their start spans with the delexicalised model, compared to 88.22% for lexicalised predicates and 91.65% for the full parsing model.

Table 5.4 reports test set results for various evaluation metrics. Start EDM is calculated by requiring only the starts of alignment spans to match, not the ends. We compare the

Model	EDM	EDM$_P$	EDM$_A$	Start EDM	Start EDM$_A$	Smatch
Top-down RNN	79.68	83.36	75.16	84.44	80.93	85.28
Arc-eager RNN	84.16	87.54	80.10	87.81	85.61	86.69
ACE	89.64	92.08	86.77	91.91	89.28	93.50

Table 5.4: DMRS parsing test set results, comparing the standard top-down attention-based and arc-eager stack-based RNN models to the grammar-based ACE parser.

Model	EDM	EDM$_P$	EDM$_A$	Smatch
Arc-eager RNN	85.48	88.14	82.20	86.50
ACE	89.58	91.82	86.92	93.52

Table 5.5: EDS parsing test set results.

performance of our baseline and stack-based models against ACE, the ERG-based parser.

Despite the promising performance of the model a gap remains between the accuracy of our parser and ACE. This result should be seen in the context that the test set is restricted to sentences which the ERG can parse, and will therefore *a priori* parse with high precision. EDM metrics excluding end-span prediction (Start EDM) show that our parser has relatively more difficulty in parsing end-span predictions than the grammar-based parser.

We also evaluate the speed of our model compared with ACE. For the unbatched version of our model, the stack-based parser parses 41.63 tokens per second, while the batched implementation parses 529.42 tokens per second using a batch size of 128. In comparison, the setting of ACE for which we report accuracies parses 7.47 tokens per second. By restricting the memory usage of ACE, which restricts its coverage, we see that ACE can parse 11.07 tokens per second at 87.7% coverage, and 15.11 tokens per second at 77.8% coverage.

Lastly we report results for parsing EDS (Table 5.5). The EDS parsing task is slightly simpler than DMRS, due to the absence of rich argument labels and additional graph edges (including undirected edges) that allow the recovery of full MRS. We see that for ACE the accuracies are very similar, while for our model EDS parsing is comparatively more accurate on the EDM metrics. We hypothesise that most of the extra information in DMRS can be obtained through the ERG, to which ACE has access but our model has not.

A qualitative analysis shows that our parser produces reasonable parses for sentences

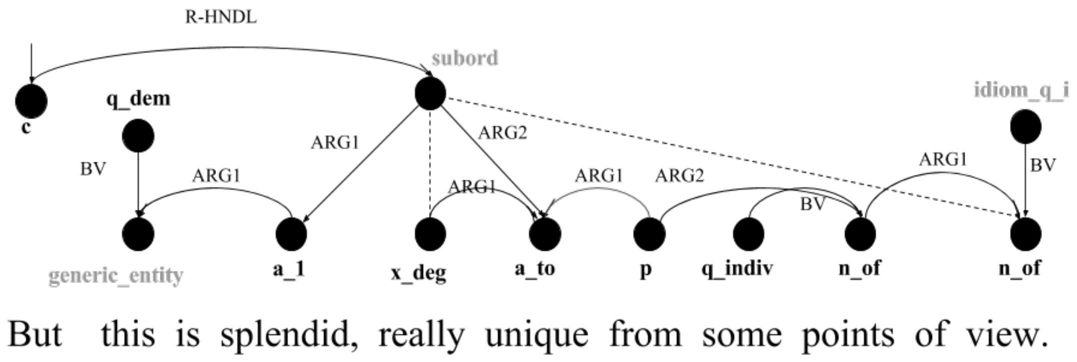


Figure 5.8: EDS parse produced by our parser for a sentence that cannot be parsed by the ERG. The unlexicalised form of the graph is shown. The span of the subordination node is indicated, other nodes align to single tokens.

Model	Concept F1	Smatch
Top-down attention, no tags	68.23	56.05
Top-down attention, with tags	70.16	57.95
Top-down pointer, lexicalised	71.25	59.39
Top-down pointer, delexicalised	72.62	59.88
Arc-eager hard attention, delexicalised	76.83	59.83
Arc-eager stack-based, delexicalised	77.93	61.21

Table 5.6: Development set results for AMR parsing (`LDC2015E86`).

that ERG cannot parse: See Figure 5.8 for an example of a sentence that ACE couldn't parse, together with the graph predicted by our model.

5.6.4 AMR parsing

Next we report results for AMR parsing, starting with the `LDC2015E86` corpus, on which most existing AMR parsers have been evaluated. Development set results on our models are given in Table 5.6. We compare two versions of the attention-based baseline: The first takes only words as input, the second embeds named entity and POS tags as well, and utilises pre-trained word embeddings. The arc-eager-based models again obtain the best performance, mainly due to improved concept prediction accuracy. However, concept prediction remains an important weakness of the model; Damonte et al. (2017) reports that state-of-the-art AMR parsers score 83% on concept prediction.

We report test set results in Table 5.7. Our best neural model outperforms the baseline

Model	Smatch
Flanigan et al. (2014)	56
Wang et al. (2016)	66.54
Damonte et al. (2017)	64
Peng and Gildea (2016)	55
Peng et al. (2017b)	52
Barzdins and Gosko (2016)	43.3
Konstas et al. (2017) (baseline)	55.5
Konstas et al. (2017) (semi-supervised)	62.1
Top-down RNN no pointers	56.56
Arc-eager RNN stack-based, delexicalised	60.11

Table 5.7: AMR parsing `LDC2015E86` test set results (Smatch F1 scores). Published results follow the number of decimals which were reported.

Model	Smatch
Top-down soft attention, no tags	54.60
Top-down soft attention, with tags	57.27
Top-down pointer, lexicalised	57.99
Top-down pointer, delexicalised	59.18

Table 5.8: Development set results for the Bio AMR corpus.

JAMR parser (Flanigan et al., 2014), but still lags behind the performance of state-of-the-art AMR parsers such as CAMR (Wang et al., 2016) and AMR Eager (Damonte et al., 2017). These models make extensive use of external resources, including syntactic parsers and semantic role labellers. Our attention-based encoder-decoder model already outperforms previous sequence-to-sequence AMR parsers (Barzdins and Gosko, 2016; Peng et al., 2017b), and the arc-eager model boosts accuracy even further. Our model also outperforms a Synchronous Hyperedge Replacement Grammar model (Peng and Gildea, 2016) whose accuracy is comparable to ours as it does not make extensive use of external resources. Our baseline is slightly more accurate than the baseline of Konstas et al. (2017), who concurrently proposed neural sequence-to-sequence AMR parsers. Konstas et al. (2017) obtain further improvements through self-training on large unlabelled corpora; this approach is complementary to the model architecture and linearisations we propose.

We also train an AMR parser for the biomedical domain. During mini-batch SGD training we sample Bio AMR sentences with a weight of 7 to each LDC sentence to balance the two sources in sampled mini-batches. We compare the performance of different RNN architectures on this dataset as well (Table 5.8).

We see that the same broad trends hold as on `LDC2015E86`, with the pointer-based models outperforming those with attention only. POS and NE embeddings give a substantial improvement. The performance of the baseline with richer embeddings is similar to that of the first pointer-based model. The main difference between these two models is that the latter uses pointers to predict constants; therefore in this case the gain due to this improved generalisation is relatively small. The delexicalised representation with separate lemma prediction improves accuracy by 1.2%. On the Bio AMR test set the best top-down model obtains 59.27% F1.

5.7 Conclusion

We advanced the state of parsing by employing deep learning techniques to parse sentences to linguistically expressive semantic representations that have not previously been parsed in an end-to-end fashion. We formulated the problem of end-to-end semantic graph parsing, and proposed a transition system that can predict these graphs, as well as graph linearisations. Our key contribution is the first robust, wide-coverage parser for MRS. Our parser is scalable, faster than existing parsers and amenable to batch processing. This work opens many opportunities for the application of linguistically expressive, structured semantic representations to natural language understanding and generation tasks. We believe that there are many future avenues to explore to further increase the speed and accuracy of such parsers. We propose several extensions to this work in the next chapter.

Chapter 6

Conclusion

6.1 Summary and conclusions

Our thesis was that distributed representations learned by neural networks and linguistically-motivated structured representations are complementary in models that form the basis of language generation and understanding applications. In this section we summarise our main results and evaluate the evidence we found in support of this thesis.

6.1.1 Generative syntactic parsing

In the first part of this thesis (Chapters 3 and 4) we proposed a series of generative dependency parsers. We showed that they can achieve accuracies up to 90.7% UAS, which is higher than many previous discriminative parsers and most generative parsers, but still less accurate than the best discriminative models. Our decoding algorithm based on particle-filtering enables more efficient parsing than previous generative models - with a small loss in accuracy we can parse up to 200 sentences per second. However for practical application in generation tasks further efficiency improvements may be required.

We showed that syntactic structure improves language modelling performance by a large margin over n-gram language models (at least for small datasets). Our models are able to adapt the syntactic structure to learn trees that are more suited to the language modelling objective. Our particle filtering-based inference approach shows that the distribution over parse trees learned by the model is very peaked, and our method gives a good estimate of the marginal distribution. While we were able to do exact inference with the RNN-based model, improving language modelling performance over RNNs is a greater challenge here.

Exact inference does however enable us to perform end-to-end learning by marginalising over the parse trees.

We found that neural networks and syntactic structure are complementary for language models based on Markov assumptions. The neural network-based generative models are also better parsers than the Bayesian model with carefully constructed priors. However, RNN language models without explicit syntactic structure outperform these models, and we were not able to show that syntactic structure leads to an improvement over RNN-only language modelling. One potential explanation is that the model structure, chosen to enable exact inference, does not capture enough syntactic information, as it only allows conditioning on one syntactic element (the stack top) at a time, while the rest of the structure has to be captured by the LSTM. Recurrent neural network grammars (Dyer et al., 2016) do not have this limitation as the model learns compositional representations over all subtrees in the stack and explicitly encodes the stack, at the cost that exact inference is not possible.

6.1.2 Semantic graph parsing

We proposed a robust, fast, and reasonably accurate MRS parser. Our approach overcomes the major disadvantage of grammar-based parsers which are not able to parse all sentences, an important requirement for applying parsers to downstream language understanding tasks. Our parser is also much faster. It is less accurate than the grammar-based approach, but the test set available for evaluation is restricted to sentences which the grammar can parse, which means that *a priori* these sentences can be parsed with high precision by the grammar-based parser. Therefore this evaluation does not fully test the generalisation ability of our model.

We showed that the same model improves the accuracy of AMR parsing over previous neural encoder-decoder approaches by a large margin. Furthermore the model is more accurate than other AMR parsers using comparable resources.

The key technical contribution of this work is to show how to perform complex structure prediction – parsing sentences to unrestricted graphs with a many-to-many relation between words and graph nodes – using RNNs. These structures could not be parsed directly (without intermediate syntax) or robustly by previous approaches. The specific contributions we made are:

1. We showed that predicting top-down linearisations of the spanning trees of graphs with attention-based encoder-decoder RNNs is a strong baseline for AMR and MRS parsing.

2. Transition-based parsing – incrementally predicting transition actions which generate the graph – outperforms the top-down generation model. The innovation here is that the transition system (stack and buffer) is decoupled from the input sentence; the decoder is formulated as a generative model over graphs, conditioned on the input sentence through an attention mechanism.

3. In addition to the graph structure, node-to-token alignments, including phrasal alignments of longer spans, are also predicted: We showed that using pointer networks to perform this prediction is more accurate than predicting alignments as strings as part of the linearised output.

4. We showed that hard attention with supervised alignments outperforms soft attention, while adding an alignment based on the top of the stack to the conditioning context further improves performance. Because the RNN encodes the entire context, the inclusion of additional features here does not affect any independence assumptions, but rather changes the inductive bias of the model to explicitly model non-sequential structure, in this case expressed through the transition system stack, which helps to overcome the recency bias of RNNs.

5. We presented a GPU implementation of the RNN parser which supports batch processing. The result is an efficient parser which is much faster than the existing grammar-based approach. However, this implementation comes at a considerable engineering cost. In particular, TensorFlow is based on static computation graphs to implement neural networks, which means that the size of all vectors have to be fixed in advanced. Therefore representing the state of the transition system is non-trivial and involves implementing a number of custom operations on Tensor data structures. Recent advances in neural network packages based on dynamic computation graphs,

including PyTorch and DyNet,[1] simplify the implementation of these models considerably, but might be less efficient.

We showed that the MRS parser is able to predict plausible structures for sentences that could not be parsed by the underlying grammar (which parsed all training sentences). Although we did not do a formal analysis, the DMRS parses produced seem mostly well-formed and consistent with the grammar, even though this is not explicitly enforced. It is plausible that the graphs could be converted back to well-formed original MRS with logical variables and scope under-specification.

The MRS parser is accurate enough that it could reasonably be used for downstream applications, in contrast to many other linguistically expressive semantic frameworks, including AMR, which are far less accurate. One of the main reasons for this is that the MRS annotations are consistent. This is due to the principles of compositional semantics, where the semantic structure represents sentence meaning rather than speaker meaning (which is more subjective and ambiguous) and the annotation process which enforces consistency with the grammar (Bender et al., 2015). In contrast, AMR has been shown to have low annotator agreement, which is at least partly due to it not being constrained by the principles of compositional semantics.

6.2 Future work

We discuss a number of opportunities for future work opened by the research in this thesis.

6.2.1 Generative syntactic models

- While our generative parsers are more accurate than most previous generative parsers, we believe there are still opportunities for further improvements in accuracy, using a combination of more expressive models and better inference methods. For example an RNN-based model with a larger conditioning context and approximate rather than exact inference could improve performance.

- RNN language models have been shown to have limited ability to predict syntax-sensitive long-distance dependencies such as subject-verb agreement (Linzen et al.,

[1] `http://dynet.io`

2016). Future work should explore whether our syntactic language models are able to overcome this limitation, and if we can learn tree structures that capture these dependencies without annotated trees, with or without agreement supervision.

- Training models for multiple languages should be investigated. We performed preliminary experiments which showed considerable variation in performance: For some languages accuracy is close to that of published results for comparable discriminative models, while for others there is a larger gap. However, model hyperparameters will have to be tuned separately for different languages. Our models are currently restricted to projective dependencies, so training for languages with a high proportion of non-projective edges will require extensions. Character-based embeddings will probably be required for morphologically-rich languages.
- We showed that semi-supervised learning can improve language modelling perplexities when training with Viterbi EM. However, alternative training methods could further improve both parsing and language modelling performance. Parsing unlabelled text with an accurate discriminative model and then learning the generative model from that could improve performance. This could be placed in an auto-encoder framework where the sentence is encoded as a parse tree.
- As generative models have been shown to have a lower sample complexity than corresponding discriminative models (Ng and Jordan, 2002), we believe that supervised generative models could be of particular use for parsing languages where only small training sets are available. Semi-supervised learning could further improve performance in low-resource settings by learning a better data distribution from unlabelled corpora.
- Further research is required into large-scale unsupervised learning of syntactic generative models in the context of language modelling or conditional generation tasks. There are still computational challenges in scaling training to large corpora. Another setup is to perform unsupervised syntactic learning in the context of an end task such as machine translation, but this will also be very expensive computationally.

- Our models can be employed in downstream language generation applications, in particular as a component of neural decoders. This could be beneficial for generating syntactically well-formed output in machine translation and other generation tasks. In particular this could be useful in low-resource tasks where neural models might not have received enough training data to produce syntactically well-formed output. A disadvantage of our generative models is that they require inference over syntactic structure during decoding, which may slow down performance. However our recurrent generative parsers learn sequential RNNs that have syntactically-informed representations due to being used to make parsing decisions. These representations could be used as the first layer in a downstream decoder to convey at least some of the syntactic information learned by the generative model.

6.2.2 Semantic parsing

- A major strand of semantic parsing research has been concerned with parsing questions to logical forms or query languages that can be executed against a database. A disadvantage of the current approach is that parsers are usually trained on domain-specific corpora and training sets are relatively small. CCG has been used as a framework for learning compositional semantic representations within the context of an end task (in this case learning a query that yields the correct answer when executed against a database). An alternative would be to use the semantic representations learned by our DMRS parser. A mapping could be learned with minimal supervision from the DMRS graph or a scope-underspecified logical form that can be obtained from DMRS graphs.

- Semantic graphs can be used to train general purpose sentence encoders, encoding sentences into fixed-sized vectors. This is of interest as it is not clear if the objective functions of existing methods to learn sentence embeddings (Kiros et al., 2015) are well-matched with representing sentence meaning. Sentence representations can be constructed from semantic graphs through recursive or convolutional neural networks.

- Recently machine reading tasks which involve being able to answer questions from documents have become prominent due to the development of large-scale datasets and the ability of neural networks to learn to encode and reason over long contexts (Hermann et al., 2015). Semantic graphs can represent complex relations which sequential encoders might fail to identify. Therefore one option would be to use our MRS parser to parse documents, and then either encode the graphs into sentence vectors or use an attention mechanism over the graph structure to perform inference.

- One of the main advantages of learning structured representations is that they can be applied to inference and reasoning: The representations can be manipulated with well-studied algorithms for graphs and logical structures. An example would be to perform logical derivations which can be used to assert the truth of a statement. Neural models also enable the continuous manipulation of these structures, for example through semantic graph encodings and attention mechanisms.

- The encoder-decoder framework can easily be extended to multi-task learning for parsing multiple semantic representations. Apart from AMR and MRS-based graphs, there are other semantic graph-based representations that can be put in the same framework, for example CCG semantic dependencies (Kuhlmann and Oepen, 2016). There are also a number of related semantic analysis tasks that produce representations that are not full-sentence semantics but can be represented in the same framework. This includes semantic role labelling (Gildea and Jurafsky, 2002), frame-semantic parsing (Das et al., 2014) and relation extraction tasks. Multi-task learning has recently been shown to be successful for semantic dependency parsing (Peng et al., 2017a).

- The role of syntax in semantic graph parsing should be explored further. In traditional semantic analysis, the syntax is predicted jointly with, and used to constrain possible semantic structures. Recent work in semantic role labelling has shown that neural models can outperform previous methods without using syntax, but that there could still be a role for syntactic structure in further improving performance (He et al., 2017). In the context of MRS parsing, the information in the ERG about syntactic structure and fine-grained morphosyntactic features may contribute to increased accuracy and well-formedness of semantic analyses.

- An advantage of parsing DMRS over other, simplified MRS-based representations is that it can be converted back to original MRS, which encodes logical forms with under-specified scope. The question is whether the MRS graphs that our parser predicts will be well-formed, satisfying the constraints of the grammar as well as allowing scope resolution. If the graphs are not always completely well-formed (as seems likely), can we incorporate more information from the grammar during training to encourage well-formedness, and specify constraints that can be enforced during incremental decoding? In particular, the ERG specifies the set of arguments each predicate is licensed to take. Scope-resolved logical forms can be particularly valuable in downstream applications.

- While we focussed our MRS parsing on newspaper text, there are also annotated corpora in other domains. It would be interesting to evaluate the performance of our neural approach on these domains, compared to the grammar-based approach. There are also treebanks or grammars for MRS in other languages, including Japanese and German, for which parsers could be trained. The coverage of these grammars are much lower than for English, so a robust parser could be particularly valuable, and it would provide a further test-bed to see how well our models can generalise outside the available grammar.

- Our approach requires a number of pre-processing steps currently performed with external tools. In a full end-to-end model this should all be performed by the neural network. One approach would be to let the input be the untokenised character sequence of the sentence. The RNN will first predict the word boundaries of tokens, excluding semantically void words. The recognized tokens can be encoded into vectors that can be used to predict word lemmas, and fed into the encoder-decoder architecture we proposed. POS tags and named entities can be predicted jointly during training so that the model can learn encoder representations that contain the information gained from this prediction, which can be leveraged to predict the semantic graphs during decoding without performing explicit tagging.

- Our model uses hard attention without any restriction on the order of the alignments of predicted predicates. If we assume that the alignment order is monotone, the model can be modified to choose from a fixed number of tokens to align to next, or to make a sequence of binary decisions on whether to generate a predicate for the current node or not. This will improve the decoding run-time, as the attention weights currently have to be computed over all input words at each output step.

- Methods to predict non-planar dependencies should be explored further. While our method can predict unbounded non-planar dependencies, this comes at the cost of having limited conditioning information available to make the prediction. An alternative would be to limit the parser to a restricted class of semantic graphs (for example 2-planar dependency graphs (Gómez-Rodríguez and Nivre, 2010)) that have high coverage on available datasets, and then focus on improving the accuracy of the non-planar edges in those graphs.

- While we focussed on the parsing problem, the ERG is a bidirectional grammar – it can be applied to both parsing and generation. Recent work has proposed neural models for generating from AMR (Konstas et al., 2017), which could be extended to generating from MRS-based graphs. This could be valuable for generation from structured information, as well as for machine translation.

- This work opens interesting opportunities for semi-supervised learning. In particular the ERG can produce candidate parses for sentences in unlabelled corpora it can parse, which will have high precision. These candidate parses can be utilised for semi-supervised learning by restricting candidate semantic graphs when performing inference over unlabelled sentences, thereby reducing the variance of the distribution. In particular this can be done in the framework of neural variational autoencoders (Miao and Blunsom, 2016).

Bibliography

Martín Abadi, 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, Dan 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. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015. Software available from tensorflow.org.

Omri Abend and Ari Rappoport. Universal Conceptual Cognitive Annotation (UCCA). In *Proceedings of ACL*, pages 228–238, Sofia, Bulgaria, August 2013.

Omri Abend and Ari Rappoport. The State of the Art in Semantic Representation. In *Proceedings of ACL*, pages 77–89, Vancouver, Canada, 2017.

Alfred V Aho, Ravi Sethi, and Jeffrey D Ullman. *Compilers: Principles, Techniques and Tools*. Addison Wesley, 1986.

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. Globally Normalized Transition-Based Neural Networks. In *Proceedings of ACL*, pages 2442–2452, Berlin, Germany, August 2016.

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Learning to Compose Neural Networks for Question Answering. In *Proceedings of NAACL*, pages 1545–1554, June 2016.

Christophe Andrieu, Arnaud Doucet, and Roman Holenstein. Particle Markov chain Monte Carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 72(3):269–342, 2010.

Yoav Artzi and Luke Zettlemoyer. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics*, 1:49–62, 2013.

Yoav Artzi, Kenton Lee, and Luke Zettlemoyer. Broad-coverage CCG Semantic Parsing with AMR. In *Proceedings of EMNLP*, pages 1699–1710, Lisbon, Portugal, September 2015.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *Proceedings of ICLR*, 2015.

Paul Baltescu and Phil Blunsom. Pragmatic Neural Language Modelling in Machine Translation. In *Proceedings of NAACL*, pages 820–829, 2015.

Paul Baltescu, Phil Blunsom, and Hieu Hoang. OxLM: A Neural Language Modelling Framework for Machine Translation. *The Prague Bulletin of Mathematical Linguistics*, 102(1):81–92, October 2014.

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. Abstract Meaning Representation for Sembanking. In *Proceedings of the Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186, Sofia, Bulgaria, August 2013.

Guntis Barzdins and Didzis Gosko. RIGA at SemEval-2016 Task 8: Impact of Smatch Extensions and Character-Level Neural Translation on AMR Parsing Accuracy. In *Proceedings of SemEval*, pages 1143–1147, 2016.

Emily M Bender, Dan Flickinger, Stephan Oepen, Woodley Packard, and Ann Copestake. Layers of interpretation: On grammar and compositionality. In *Proceedings of the International Conference on Computational Semantics*, pages 239–249, 2015.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, 3:1137–1155, 2003.

Adam L Berger, Vincent J Della Pietra, and Stephen A Della Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71, 1996.

Daniel M. Bikel. Intricacies of Collins' Parsing Model. *Computational Linguistics*, 30(4): 479–511, 2004.

Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly Media, 2009.

Yonatan Bisk and Julia Hockenmaier. Probing the Linguistic Strengths and Limitations of Unsupervised Grammar Induction. In *Proceedings of ACL*, pages 1395–1404, Beijing,China, July 2015.

E. Black, S. Abney, S. Flickenger, C. Gdaniec, C. Grishman, P. Harrison, D. Hindle, R. Ingria, F. Jelinek, J. Klavans, M. Liberman, M. Marcus, S. Roukos, B. Santorini, and T. Strzalkowski. Procedure for Quantitatively Comparing the Syntactic Coverage of English Grammars. In *Proceedings of the Workshop on Speech and Natural Language*, pages 306–311, Pacific Grove, California, 1991.

Phil Blunsom and Trevor Cohn. Unsupervised Induction of Tree Substitution Grammars for Dependency Parsing. In *Proceedings of EMNLP*, pages 1204–1213, 2010.

Phil Blunsom and Trevor Cohn. A Hierarchical Pitman-Yor Process HMM for Unsupervised Part of Speech Induction. In *Proceedings of ACL*, pages 865–874, Portland, Oregon, USA, June 2011.

Phil Blunsom, Trevor Cohn, Sharon Goldwater, and Mark Johnson. A Note on the Implementation of Hierarchical Dirichlet Processes. In *Proceedings of ACL-IJCNLP*, pages 337–340, 2009.

Alena Böhmová, Jan Hajič, Eva Hajičová, and Barbora Hladká. *The Prague Dependency Treebank*, pages 103–127. Springer Netherlands, Dordrecht, 2003.

Andrew Borthwick. *A maximum entropy approach to named entity recognition*. PhD thesis, New York University, 1999.

Jan A. Botha and Phil Blunsom. Compositional Morphology for Word Representations and Language Modelling. In *Proceedings of the International Conference on Machine Learning*, pages 1899–1907, Beijing, China, 2014.

Léon Bottou. Online Algorithms and Stochastic Approximations. In David Saad, editor, *Online Learning and Neural Networks*, pages 9–42. Cambridge University Press, 1998.

Samuel R. Bowman, Jon Gauthier, Abhinav Rastogi, Raghav Gupta, Christopher D. Manning, and Christopher Potts. A Fast Unified Model for Parsing and Sentence Understanding. In *Proceedings of ACL*, pages 1466–1477, Berlin, Germany, August 2016.

Peter F. Brown, Peter V. deSouza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai. Class-based N-gram Models of Natural Language. *Computational Linguistics*, 18 (4):467–479, 1992.

Peter F Brown, Vincent J Della Pietra, Stephen A Della Pietra, and Robert L Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational linguistics*, 19(2):263–311, 1993.

Sabine Buchholz and Erwin Marsi. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of CoNLL*, pages 149–164. 2006.

Jan Buys and Phil Blunsom. Generative Incremental Dependency Parsing with Neural Networks. In *Proceedings of ACL-IJCNLP (Short Papers)*, pages 863–869, 2015a.

Jan Buys and Phil Blunsom. A Bayesian Model for Generative Transition-based Dependency Parsing. In *Proceedings of the International Conference on Dependency Linguistics*, pages 58–67, 2015b.

Jan Buys and Phil Blunsom. Robust Incremental Neural Semantic Graph Parsing. In *Proceedings of ACL*, pages 1215–1226, 2017a.

Jan Buys and Phil Blunsom. Oxford at SemEval-2017 Task 9: Neural AMR Parsing with Pointer-Augmented Attention. In *Proceedings of SemEval*, pages 914–919, 2017b.

Jan Buys and Phil Blunsom. Neural Syntactic Generative Models with Exact Marginalization. In *Proceedings of NAACL*, 2018. To appear.

Shu Cai and Kevin Knight. Smatch: An Evaluation Metric for Semantic Feature Structures. In *Proceedings of ACL (Short Papers)*, pages 748–752, 2013.

Ulrich Callmeier. PET - a platform for experimentation with efficient HPSG processing techniques. *Natural Language Engineering*, 6(1):99–107, 2000.

John Canny, David Hall, and Dan Klein. A Multi-Teraflop Constituency Parser using GPUs. In *Proceedings of EMNLP*, pages 1898–1907. October 2013.

Xavier Carreras. Experiments with a higher-order projective dependency parser. In *Proceedings of EMNLP-CoNLL*, pages 957–961, 2007.

Angel X Chang and Christopher D Manning. SUTime: A library for recognizing and normalizing time expressions. In *Proceedings of the International Conference on Language Resources and Evaluation*, pages 3735–3740, 2012.

Eugene Charniak. Immediate-head parsing for language models. In *Proceedings of ACL*, pages 124–131. 2001.

Eugene Charniak and Mark Johnson. Coarse-to-Fine n-Best Parsing and MaxEnt Discriminative Reranking. In *Proceedings of ACL*, pages 173–180, 2005.

Ciprian Chelba and Frederick Jelinek. Structured language modeling. *Computer Speech and Language*, 14(4):283–332, 2000.

Danqi Chen and Christopher D Manning. A Fast and Accurate Dependency Parser using Neural Networks. In *Proceedings of EMNLP*, pages 740–750, 2014.

Stanley F. Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. *Computer Speech and Language*, 13(4):359–394, 1999.

Do Kook Choe and Eugene Charniak. Parsing as Language Modeling. In *EMNLP*, pages 2331–2336, 2016.

Jinho D. Choi and Andrew McCallum. Transition-based dependency parsing with selectional branching. In *Proceedings of ACL*, pages 1052–1062, 2013.

Noam Chomsky. *Syntactic Structures*. Mouton, 1957.

Noam Chomsky. On Certain Formal Properties of Grammars. *Information and Control*, 2: 137–167, 1959.

Noam Chomsky. *Aspects of the Theory of Syntax*. The MIT Press, Cambridge, 1965.

Noam Chomsky. *Lectures on Government and Binding*. Foris, Dordrecht, 1981.

Shay B Cohen, Kevin Gimpel, and Noah A Smith. Logistic normal priors for unsupervised probabilistic grammar induction. In *Advances in Neural Information Processing Systems*, pages 321–328, 2008.

Shay B. Cohen, Carlos Gómez-Rodríguez, and Giorgio Satta. Exact Inference for Generative Probabilistic Non-Projective Dependency Parsing. In *Proceedings of EMNLP*, pages 1234–1245, 2011.

Michael Collins. Three Generative, Lexicalised Models for Statistical Parsing. In *Proceedings of ACL*, pages 16–23, 1997.

Michael Collins and Brian Roark. Incremental Parsing with the Perceptron Algorithm. In *Proceedings of ACL*, pages 111–118, Barcelona, Spain, July 2004.

Ann Copestake. Invited Talk: Slacker Semantics: Why Superficiality, Dependency and Avoidance of Commitment can be the Right Way to Go. In *Proceedings of EACL*, pages 1–9, Athens, Greece, March 2009.

Ann Copestake, Dan Flickinger, Rob Malouf, Susanne Riehemann, and Ivan Sag. Translation using Minimal Recursion Semantics. In *Proceedings of the International Conference on Theoretical and Methodological Issues in Machine Translation*, pages 15–32, 1995.

Ann Copestake, Dan Flickinger, Carl Pollard, and Ivan A Sag. Minimal recursion semantics: An introduction. *Research on Language and Computation*, 3(2-3):281–332, 2005.

Ann Copestake, Guy Emerson, Michael Wayne Goodman, Matic Horvat, Alexander Kuhnle, and Ewa Muszyska. Resources for building applications with Dependency Minimal Recursion Semantics. In *Proceedings of the International Conference on Language Resources and Evaluation*, pages 1240–1247, Portoro, Slovenia, 2016.

James Cross and Liang Huang. Incremental Parsing with Minimal Features Using Bi-Directional LSTM. In *Proceedings of ACL*, page 32, 2016.

Marco Damonte, Shay B. Cohen, and Giorgio Satta. An Incremental Parser for Abstract Meaning Representation. In *Proceedings of EACL*, pages 536–546, April 2017.

Dipanjan Das, Desai Chen, André FT Martins, Nathan Schneider, and Noah A Smith. Frame-semantic parsing. *Computational linguistics*, 40(1):9–56, 2014.

Hal Daumé III. Unsupervised search-based structured prediction. In *Proceedings of ICML*, pages 209–216. 2009.

Marie-Catherine De Marneffe and Christopher D Manning. The Stanford typed dependencies representation. In *Proceedings of the Workshop on Cross-Framework and Cross-Domain Parser Evaluation*, pages 1–8, 2008.

Li Dong and Mirella Lapata. Language to Logical Form with Neural Attention. *CoRR*, abs/1601.01280, 2016.

Arnaud Doucet and Adam M Johansen. A tutorial on particle filtering and smoothing: Fifteen years later. *Handbook of nonlinear filtering*, 12(656-704):3, 2009.

Timothy Dozat and Christopher D. Manning. Deep Biaffine Attention for Neural Dependency Parsing. In *Proceedings of ICLR*, 2017.

Rebecca Dridan and Stephan Oepen. Parser evaluation using elementary dependency matching. In *Proceedings of the International Conference on Parsing Technologies*, pages 225–230. 2011.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12: 2121–2159, July 2011.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. Transition-Based Dependency Parsing with Stack Long Short-Term Memory. In *Proceedings of ACL*, pages 334–343, Beijing, China, July 2015.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. Recurrent neural network grammars. In *Proceedings of NAACL*, pages 199–209, 2016.

Jason Eisner. Three New Probabilistic Models for Dependency Parsing: An Exploration. In *Proceedings of COLING*, pages 340–345, 1996.

Jason Eisner. Inside-Outside and Forward-Backward Algorithms Are Just Backprop (tutorial paper). In *Proceedings of the Workshop on Structured Prediction for NLP*, pages 1–17, Austin, TX, November 2016.

Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

Ahmad Emami and Frederick Jelinek. A Neural Syntactic Language Model. *Machine Learning*, 60(1-3):195–227, September 2005.

Martin B.H. Everaert, Marinus A.C. Huybregts, Noam Chomsky, Robert C. Berwick, and Johan J. Bolhuis. Structures, Not Strings: Linguistics as Part of the Cognitive Sciences. *Trends in Cognitive Sciences*, 19(12):729 – 743, 2015.

Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling. In *Proceedings of ACL*, pages 363–370. 2005.

Jeffrey Flanigan, Sam Thomson, Jaime G. Carbonell, Chris Dyer, and Noah A. Smith. A Discriminative Graph-Based Parser for the Abstract Meaning Representation. In *Proceedings of ACL*, pages 1426–1436, 2014.

Dan Flickinger. On building a more effcient grammar by exploiting types. *Natural Language Engineering*, 6(01):15–28, 2000.

Dan Flickinger, Yi Zhang, and Valia Kordoni. DeepBank. A dynamically annotated treebank of the Wall Street Journal. In *Proceedings of the International Workshop on Treebanks and Linguistic Theories*, pages 85–96, 2012.

William R Foland Jr and James H Martin. CU-NLP at SemEval-2016 task 8: AMR parsing using LSTM-based recurrent neural networks. In *Proceedings of SemEval*, pages 1197–1201, 2016.

Daniel Gildea and Daniel Jurafsky. Automatic labeling of semantic roles. *Computational Linguistics*, 28(3):245–288, 2002.

Yoav Goldberg and Joakim Nivre. A Dynamic Oracle for Arc-Eager Dependency Parsing. In *Proceedings of COLING 2012*, pages 959–976, Mumbai, India, December 2012.

Yoav Goldberg and Joakim Nivre. Training Deterministic Parsers with Non-Deterministic Oracles. *Transactions of the Association for Computational Linguistics*, 1:403–414, 2013.

Sharon Goldwater, Thomas L. Griffiths, and Mark Johnson. Contextual Dependencies in Unsupervised Word Segmentation. In *Proceedings of COLING-ACL*, pages 673–680, Sydney, Australia, July 2006a.

Sharon Goldwater, Mark Johnson, and Thomas L. Griffiths. Interpolating between types and tokens by estimating power-law generators. In Y. Weiss, P. B. Schölkopf, and J. C. Platt, editors, *Advances in Neural Information Processing Systems*, pages 459–466. 2006b.

Carlos Gómez-Rodríguez and Joakim Nivre. A Transition-Based Parser for 2-Planar Dependency Structures. In *Proceedings of ACL*, pages 1492–1501, Uppsala, Sweden, July 2010.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014.

Joshua Goodman. Classes for fast maximum entropy training. In *Proceedings of ICASSP*, pages 561–564, 2001.

N. J. Gordon, D. J. Salmond, and A. F. M. Smith. Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *IEEE Proceedings F - Radar and Signal Processing*, 140(2):107–113, April 1993.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *CoRR*, abs/1410.5401, 2014.

Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. Learning to Transduce with Unbounded Memory. In *Advances in Neural Information Processing Systems*, pages 1828–1836, 2015.

Johan Hall, Joakim Nivre, and Jens Nilsson. Discriminative Classifiers for Deterministic Dependency Parsing. In *Proceedings of the COLING-ACL*, pages 316–323, Sydney, Australia, July 2006.

Luheng He, Kenton Lee, Mike Lewis, and Luke Zettlemoyer. Deep Semantic Role Labeling: What Works and What's Next. In *Proceedings of ACL*, pages 473–483, Vancouver, Canada, July 2017.

William P. Headden, III, Mark Johnson, and David McClosky. Improving Unsupervised Dependency Parsing with Richer Contexts and Smoothing. In *Proceedings of NAACL*, pages 101–109, Boulder, Colorado, 2009.

James Henderson. Neural network probability estimation for broad coverage parsing. In *Proceedings of EACL*, pages 131–138. 2003a.

James Henderson. Inducing History Representations for Broad Coverage Statistical Parsing. In *Proceedings of NAACL*, pages 24–31, Edmonton, Canada, 2003b.

James Henderson. Discriminative training of a neural network statistical parser. In *Proceedings of ACL*, pages 95–102. 2004.

Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*, pages 1693–1701, 2015.

Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

Julia Hockenmaier and Mark Steedman. CCGbank: a corpus of CCG derivations and dependency structures extracted from the Penn Treebank. *Computational Linguistics*, 33(3):355–396, 2007.

Liang Huang and Kenji Sagae. Dynamic Programming for Linear-Time Incremental Parsing. In *Proceedings of ACL*, pages 1077–1086, 2010.

Liang Huang, Suphan Fayong, and Yang Guo. Structured Perceptron with Inexact Search. In *Proceedings of NAACL*, pages 142–151, Montréal, Canada, June 2012.

Robin Jia and Percy Liang. Data Recombination for Neural Semantic Parsing. In *Proceedings of ACL*, pages 12–22, Berlin, Germany, August 2016.

Richard Johansson and Pierre Nugues. Extended constituent-to-dependency conversion for English. In *Proceedings of the Nordic Conference of Computational Linguistics*, pages 105–112, Tartu, Estonia, 2007.

Mark Johnson. Parsing in Parallel on Multiple Cores and GPUs. In *Proceedings of the Australasian Language Technology Association Workshop*, pages 29–37, Canberra, Australia, December 2011.

Michael I Jordan. Serial Order: A Parallel Distributed Processing Approach. Technical report, DTIC Document, 1986.

Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In *Proceedings of the International Conference on Machine Learning*, pages 2342–2350, 2015.

Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the Limits of Language Modeling. *CoRR*, abs/1602.02410, 2016.

Ronald M Kaplan and Joan Bresnan. Lexical-functional grammar: A formal system for grammatical representation. *Formal Issues in Lexical-Functional Grammar*, pages 29–130, 1982.

Andrej Karpathy, Justin Johnson, and Li Fei-Fei. Visualizing and understanding recurrent networks. *CoRR*, abs/1506.02078, 2015.

Chloé Kiddon, Luke Zettlemoyer, and Yejin Choi. Globally Coherent Text Generation with Neural Checklist Models. In *Proceedings of EMNLP*, pages 329–339, 2016.

Yoon Kim, Carl Denton, Luong Hoang, and Alexander M. Rush. Structured Attention Networks. In *Proceedings of ICLR*, 2017.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *Proceedings of ICLR*, 2015.

Eliyahu Kiperwasser and Yoav Goldberg. Simple and Accurate Dependency Parsing Using Bidirectional LSTM Feature Representations. *Transactions of the Association for Computational Linguistics*, 4:313–327, 2016.

Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Skip-thought vectors. In *Advances in neural information processing systems*, pages 3294–3302, 2015.

Dan Klein and Christopher D. Manning. Accurate Unlexicalized Parsing. In *Proceedings of ACL*, pages 423–430, 2003.

Dan Klein and Christopher D Manning. Corpus-based induction of syntactic structure: Models of dependency and constituency. In *Proceedings of ACL*, pages 478–586, 2004.

Reinhard Kneser and Hermann Ney. Improved backing-off for m-gram language modeling. In *Proceedings of ICASSP*, volume 1, pages 181–184. 1995.

Donald E Knuth. On the translation of languages from left to right. *Information and control*, 8(6):607–639, 1965.

Lingpeng Kong and Noah A Smith. An empirical comparison of parsing methods for Stanford dependencies. *CoRR*, abs/1404.4314, 2014.

Ioannis Konstas, Srinivasan Iyer, Mark Yatskar, Yejin Choi, and Luke Zettlemoyer. Neural AMR: Sequence-to-Sequence Models for Parsing and Generation. In *Proceedings of ACL*, pages 146–157, Vancouver, Canada, 2017.

Terry Koo and Michael Collins. Efficient third-order dependency parsers. In *Proceedings of ACL*, pages 1–11, 2010.

Terry Koo, Alexander M. Rush, Michael Collins, Tommi Jaakkola, and David Sontag. Dual decomposition for parsing with nonprojective head automata. In *Proceedings of EMNLP*, pages 1288–1298, 2010.

Marco Kuhlmann and Stephan Oepen. Towards a Catalogue of Linguistic Graph Banks. *Computational Linguistics*, 42(4):819–827, 2016.

Marco Kuhlmann, Carlos Gómez-Rodríguez, and Giorgio Satta. Dynamic Programming Algorithms for Transition-Based Dependency Parsers. In *Proceedings of ACL*, pages 673–682, 2011.

Adhiguna Kuncoro, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, Graham Neubig, and Noah A. Smith. What Do Recurrent Neural Network Grammars Learn About Syntax? In *Proceedings of EACL*, pages 1249–1258, Valencia, Spain, April 2017.

Tom Kwiatkowski, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. Inducing probabilistic CCG grammars from logical form with higher-order unification. In *Proceedings of EMNLP*, pages 1223–1233. 2010.

John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the International Conference on Machine Learning*, volume 1, pages 282–289, 2001.

Tao Lei, Yu Xin, Yuan Zhang, Regina Barzilay, and Tommi Jaakkola. Low-Rank Tensors for Scoring Dependency Structures. In *Proceedings of ACL*, pages 1381–1391, 2014.

Roger P Levy, Florencia Reali, and Thomas L Griffiths. Modeling the effects of memory on human online sentence processing with particle filters. In *Advances in neural information processing systems*, pages 937–944, 2009.

Wang Ling, Chris Dyer, Alan W Black, and Isabel Trancoso. Two/Too Simple Adaptations of Word2Vec for Syntax Problems. In *Proceedings of NAACL*, pages 1299–1304, May–June 2015.

Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. Assessing the Ability of LSTMs to Learn Syntax-Sensitive Dependencies. *Transactions of the Association for Computational Linguistics*, 4:521–535, 2016.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. The Stanford CoreNLP Natural Language Processing Toolkit. In *Proceedings of ACL: System Demonstrations*, pages 55–60, 2014.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2): 313–330, 1993.

Andre Martins, Miguel Almeida, and A. Noah Smith. Turning on the Turbo: Fast Third-Order Non-Projective Turbo Parsers. In *Proceedings of ACL (Short Papers)*, pages 617–622, Sofia, Bulgaria, 2013.

André F. T. Martins, Noah A. Smith, and Eric P. Xing. Concise Integer Linear Programming Formulations for Dependency Parsing. In *Proceedings of ACL-IJCNLP*, pages 342–350, Suntec, Singapore, 2009.

Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. Efficient HPSG Parsing with Supertagging and CFG-filtering. In *Proceedings of the International Joint Conference on Artifical Intelligence*, pages 1671–1676, Hyderabad, India, 2007.

Jonathan May. SemEval-2016 Task 8: Meaning Representation Parsing. In *Proceedings of SemEval*, pages 1063–1073, June 2016.

David McClosky, Eugene Charniak, and Mark Johnson. Effective self-training for parsing. In *Proceedings of NAACL*, pages 152–159. 2006.

Ryan Mcdonald and Fernando Pereira. Online Learning of Approximate Dependency Parsing Algorithms. In *Proceedings of EACL*, pages 81–88, 2006.

Ryan McDonald and Giorgio Satta. On the Complexity of Non-projective Data-driven Dependency Parsing. In *Proceedings of the International Conference on Parsing Technologies*, pages 121–132, Prague, Czech REpublic, 2007.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of EMNLP*, pages 523–530, 2005a.

Ryan McDonald, Slav Petrov, and Keith Hall. Multi-source transfer of delexicalized dependency parsers. In *Proceedings of EMNLP*, pages 62–72. 2011.

Ryan T. McDonald, Koby Crammer, and Fernando C. N. Pereira. Online Large-Margin Training of Dependency Parsers. In *Proceedings of ACL*, pages 91–98, 2005b.

Yishu Miao and Phil Blunsom. Language as a Latent Variable: Discrete Generative Models for Sentence Compression. In *Proceedings of EMNLP*, pages 319–328, Austin, Texas, 2016.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Proceedings of INTERSPEECH*, pages 1045–1048, 2010.

Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Extensions of recurrent neural network language model. In *Proceedings of ICASSP*, pages 5528–5531. 2011.

George A. Miller. WordNet: A Lexical Database for English. *Communications of ACM*, 38(11):39–41, November 1995.

Dipendra Kumar Misra and Yoav Artzi. Neural Shift-Reduce CCG Semantic Parsing. In *Proceedings of EMNLP*, pages 1775–1786, Austin, Texas, November 2016.

Yusuke Miyao and Jun'ichi Tsujii. Feature Forest Models for Probabilistic HPSG Parsing. *Computational Linguistics*, 34(1):35–80, 2008.

Andriy Mnih and Geoffrey Hinton. Three new graphical models for statistical language modelling. In *Proceedings of the International Conference on Machine Learning*, pages 641–648. 2007.

Radford M. Neal. Slice sampling. *The Annals of Statistics*, 31(3):705–767, 06 2003.

Andrew Y Ng and Michael I Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Advances in neural information processing systems*, pages 841–848, 2002.

Joakim Nivre. An efficient algorithm for projective dependency parsing. In *Proceedings of the International Workshop on Parsing Technologies*, pages 149–160, 2003.

Joakim Nivre. Algorithms for Deterministic Incremental Dependency Parsing. *Computational Linguistics*, 34(4):513–553, 2008.

Joakim Nivre and Mario Scholz. Deterministic Dependency Parsing of English Text. In *Proceedings of COLING*, 2004.

Joakim Nivre, Johan Hall, and Jens Nilsson. Maltparser: A data-driven parser-generator for dependency parsing. In *Proceedings of the International Conference on Language Resources and Evaluation*, volume 6, pages 2216–2219, 2006.

Joakim Nivre, Johan Hall, Sandra Kübler, Ryan T. McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. The CoNLL 2007 Shared Task on Dependency Parsing. In *Proceedings of EMNLP-CoNLL: CoNLL shared task*, pages 915–932, Prague, Czech Republic, June 2007.

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajic, Christopher D. Manning, Ryan T. McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Reut Tsarfaty, and Daniel Zeman. Universal Dependencies v1: A Multilingual Treebank Collection. In *Proceedings of the International Conference on Language Resources and Evaluation*, 2016.

Hiroshi Noji, Yusuke Miyao, and Mark Johnson. Using Left-corner Parsing to Encode Universal Structural Constraints in Grammar Induction. In *Proceedings of EMNLP*, pages 33–43, Austin, Texas, November 2016.

Stephan Oepen and Jan Tore Lønning. Discriminant-based MRS banking. In *Proceedings of the International Conference on Language Resources and Evaluation*, pages 1250–1255, 2006.

Stephan Oepen, Dan Flickinger, Kristina Toutanova, and Christopher D. Manning. LinGO Redwoods. *Research on Language and Computation*, 2(4):575–596, 2004.

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Dan Flickinger, Jan Hajic, Angelina Ivanova, and Yi Zhang. SemEval 2014 Task 8: Broad-Coverage Semantic Dependency Parsing. In *Proceedings of SemEval*, pages 63–72, Dublin, Ireland, August 2014.

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Silvie Cinkova, Dan Flickinger, Jan Hajic, and Zdenka Uresova. SemEval 2015 Task 18: Broad-Coverage Semantic Dependency Parsing. In *Proceedings of SemEval*, pages 915–926, June 2015.

Martha Palmer, Daniel Gildea, and Paul Kingsbury. The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31(1):71–106, 2005.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. *Proceedings of ICML*, 28:1310–1318, 2013.

John K Pate and Mark Johnson. Grammar induction from (lots of) words alone. In *Proceedings of COLING*, pages 23–32, Osaka, Japan, December 2016.

Hao Peng, Sam Thomson, and Noah A. Smith. Deep Multitask Learning for Semantic Dependency Parsing. In *Proceedings of ACL*, pages 2037–2048, Vancouver, Canada, July 2017a.

Xiaochang Peng and Daniel Gildea. UofR at SemEval-2016 Task 8: Learning Synchronous Hyperedge Replacement Grammar for AMR Parsing. In *Proceedings of SemEval*, pages 1185–1189, June 2016.

Xiaochang Peng, Chuan Wang, Daniel Gildea, and Nianwen Xue. Addressing the Data Sparsity Issue in Neural AMR Parsing. In *Proceedings of EACL*, pages 366–375, 2017b.

Slav Petrov and Dan Klein. Discriminative Log-Linear Grammars with Latent Variables. In *Advances in Neural Information Processing Systems*, pages 1153–1160, Vancouver, British Columbia, Canada., December 2007.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. Learning Accurate, Compact, and Interpretable Tree Annotation. In *Proceedings of COLING-ACL*, pages 433–440, 2006.

Jim Pitman. Exchangeable and partially exchangeable random partitions. *Probability Theory and Related Fields*, 102(2):145–158, Jun 1995.

Carl Pollard and Ivan A Sag. *Head-driven phrase structure grammar*. University of Chicago Press, 1994.

Michael Pust, Ulf Hermjakob, Kevin Knight, Daniel Marcu, and Jonathan May. Parsing English into Abstract Meaning Representation Using Syntax-Based Machine Translation. In *Proceedings of EMNLP*, pages 1143–1154, Lisbon, Portugal, 2015.

Pushpendre Rastogi, Ryan Cotterell, and Jason Eisner. Weighting Finite-State Transductions With Neural Context. In *Proceedings of NAACL*, pages 623–633, 2016.

Ariya Rastrow, Mark Dredze, and Sanjeev Khudanpur. Efficient Structured Language Modeling for Speech Recognition. In *Proceedings of INTERSPEECH*, pages 1660–1663, Portland, Oregon, USA, September 2012.

Adwait Ratnaparkhi. A maximum entropy model for part-of-speech tagging. In *Proceedings of EMNLP*, volume 1, pages 133–142. 1996.

Brian Roark. Probabilistic top-down parsing and language modeling. *Computational Linguistics*, 27(2):249–276, 2001.

Ronald Rosenfeld. A maximum entropy approach to adaptive statistical language modelling. *Computer Speech and Language*, 10:187–228, 1996.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by backpropagating errors. *Nature*, 323:533–536, 1986.

Kenji Sagae and Jun'ichi Tsujii. Shift-Reduce Dependency DAG Parsing. In *Proceedings of COLING*, pages 753–760, August 2008.

Tianze Shi, Liang Huang, and Lillian Lee. Fast(er) Exact Decoding and Global Training for Transition-Based Dependency Parsing via a Minimal Feature Set. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 12–23, Copenhagen, Denmark, September 2017.

Stuart M Shieber, Yves Schabes, and Fernando CN Pereira. Principles and implementation of deductive parsing. *The Journal of logic programming*, 24(1):3–36, 1995.

Noah A. Smith and Jason Eisner. Annealing Structural Bias in Multilingual Weighted Grammar Induction. In *Proceedings of COLING-ACL*, pages 569–576, Sydney, Australia, 2006.

Valentin I Spitkovsky, Hiyan Alshawi, and Daniel Jurafsky. From baby steps to Leapfrog: how Less is More in unsupervised dependency parsing. In *Proecedings of NAACL*, pages 751–759, 2010a.

Valentin I. Spitkovsky, Hiyan Alshawi, Daniel Jurafsky, and Christopher D. Manning. Viterbi Training Improves Unsupervised Dependency Parsing. In *Proceedings of CoNLL*, pages 9–17, Uppsala, Sweden, 2010b.

Mark Steedman. *The syntactic process*. MIT Press, 2000.

Sainbayar Sukhbaatar, arthur Szlam, Jason Weston, and Rob Fergus. End-To-End Memory Networks. In *Advances in Neural Information Processing Systems*, pages 2440–2448, 2015.

Ilya Sutskever, Oriol Vinyals, and Quoc VV Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112, 2014.

Ben Taskar, Dan Klein, Michael Collins, Daphne Koller, and Christopher D. Manning. Max-Margin Parsing. In *Proceedings of EMNLP*, pages 1–8, Barcelona, Spain, July 2004.

Yee Whye Teh. A Hierarchical Bayesian Language Model Based On Pitman-Yor Processes. In *Proceedings of COLING-ACL*, pages 982–992, 2006a.

Yee Whye Teh. Bayesian Interpretation of Interpolated Kneser-Ney. Technical Report TRA2/06, School of Computing, National University of Singapore, 2006b.

Ivan Titov and James Henderson. A Latent Variable Model for Generative Dependency Parsing. In *Proceedings of the International Conference on Parsing Technologies*, pages 144–155, 2007.

Ivan Titov, James Henderson, Paola Merlo, and Gabriele Musillo. Online Graph Planarisation for Synchronous Parsing of Semantic and Syntactic Dependencies. In *Proceedings of IJCAI*, pages 1562–1567, 2009.

Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network. In *Proceedings of NAACL*, pages 173–180, 2003.

Kristina Toutanova, Christopher D. Manning, Dan Flickinger, and Stephan Oepen. Stochastic HPSG Parse Disambiguation using the Redwoods Corpus. *Research on Language and Computation*, 3(1):83–105, 2005.

Ke M. Tran, Yonatan Bisk, Ashish Vaswani, Daniel Marcu, and Kevin Knight. Unsupervised Neural Hidden Markov Models. In *Proceedings of the Workshop on Structured Prediction for NLP*, pages 63–71, Austin, TX, November 2016.

Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel Recurrent Neural Networks. In *Proceedings of ICML*, pages 1747–1756, 2016.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer Networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, pages 2692–2700. 2015a.

Oriol Vinyals, Łukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. Grammar as a foreign language. In *Advances in Neural Information Processing Systems*, pages 2755–2763, 2015b.

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3156–3164, 2015c.

Hanna M Wallach, Charles Sutton, and Andrew McCallum. Bayesian modeling of dependency trees using hierarchical Pitman-Yor priors. In *ICML Workshop on Prior Knowledge for Text and Language Processing*, pages 15–20, 2008.

Chuan Wang, Nianwen Xue, and Sameer Pradhan. A transition-based algorithm for AMR parsing. In *Proceedings of NAACL*, pages 366–375, 2015a.

Chuan Wang, Nianwen Xue, and Sameer Pradhan. Boosting Transition-based AMR Parsing with Refined Actions and Auxiliary Analyzers. In *Proceedings of ACL (Short Papers)*, pages 857–862, 2015b.

Chuan Wang, Sameer Pradhan, Xiaoman Pan, Heng Ji, and Nianwen Xue. CAMR at SemEval-2016 Task 8: An Extended Transition-based AMR Parser. In *Proceedings of SemEval*, pages 1173–1178, June 2016.

David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. Structured Training for Neural Network Transition-Based Parsing. In *Proceedings of ACL*, pages 323–333, 2015.

Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *CoRR*, abs/1410.3916, 2014.

Hiroyasu Yamada and Yuji Matsumoto. Statistical dependency analysis with support vector machines. In *Proceedings of the International Conference on Parsing Technologies*, volume 3, pages 195–206, 2003.

Youngmin Yi, Chao-Yue Lai, Slav Petrov, and Kurt Keutzer. Efficient Parallel CKY Parsing on GPUs. In *Proceedings of the International Conference on Parsing Technologies*, pages 175–185, Dublin, Ireland, October 2011.

D. Yogatama, C. Dyer, W. Ling, and P. Blunsom. Generative and Discriminative Text Classification with Recurrent Neural Networks. *CoRR*, abs/1703.01898, 2017.

Dani Yogatama, Phil Blunsom, Chris Dyer, Edward Grefenstette, and Wang Ling. Learning to Compose Words into Sentences with Reinforcement Learning. *CoRR*, abs/1611.09100, 2016.

Gisle Ytrestøl. *Transition-Based Parsing for Large-Scale Head-Driven Phrase Structure Grammars*. PhD thesis, University of Oslo, 2012.

Lei Yu, Jan Buys, and Phil Blunsom. Online Segment to Segment Neural Transduction. In *Proceedings of EMNLP*, pages 1307–1316, 2016.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent Neural Network Regularization. *CoRR*, abs/1409.2329, 2014.

Luke S. Zettlemoyer and Michael Collins. Learning to Map Sentences to Logical Form: Structured Classification with Probabilistic Categorial Grammars. In *Proceedings of UAI*, pages 658–666, 2005.

Yi Zhang and Hans-Ulrich Krieger. Large-scale corpus-driven PCFG approximation of an HPSG. In *Proceedings of the International Conference on Parsing Technologies*, pages 198–208. 2011.

Yi Zhang, Stephan Oepen, and John Carroll. Efficiency in Unification-Based N-Best Parsing. In *Proceedings of the International Conference on Parsing Technologies*, pages 48–59, June 2007.

Yi Zhang, Stephan Oepen, Rebecca Dridan, Dan Flickinger, , and Hans-Ulrich Krieger. Robust Parsing, Meaning Composition, and Evaluation: Integrating Grammar Approximation, Default Unification, and Elementary Semantic Dependencies, 2014. Unpublished manuscript.

Yue Zhang and Stephen Clark. A Tale of Two Parsers: Investigating and Combining Graph-based and Transition-based Dependency Parsing. In *Proceedings of EMNLP*, pages 562–571, Honolulu, Hawaii, 2008.

Yue Zhang and Joakim Nivre. Transition-based dependency parsing with rich non-local features. In *Proceedings of ACL (Short Papers)*, pages 188–193, 2011.

Zhi-Hua Zhou and Ming Li. Tri-training: Exploiting unlabeled data using three classifiers. *IEEE Transactions on knowledge and Data Engineering*, 17(11):1529–1541, 2005.

George Kingsley Zipf. *Human behavior and the principle of least effort*. Addison-Wesley Press, 1949.