Neural language models with latent syntax

MSc Thesis *(Afstudeerscriptie)*

written by

Daan van Stigt
(born August 17, 1992 in Amsterdam, Netherlands)

under the supervision of **Dr. Wilker Aziz**, and submitted to the Board of Examiners in partial fulfillment of the requirements for the degree of

MSc in Logic

at the *Universiteit van Amsterdam*.

Date of the public defense:	**Members of the Thesis Committee:**
May 24, 2019	Dr. Caio Corro
	Prof. Dr. Khalil Sima'an
	Prof. Dr. Yde Venema

INSTITUTE FOR LOGIC, LANGUAGE AND COMPUTATION

Abstract

In this thesis I investigate the question: *What are effective ways of incorporating syntactic structure into neural language models?*

In this thesis I:

- study a class of neural language models that merges generative transition-based parsing with recurrent neural networks in order to model sentences together with their latent syntactic structure;
- propose a new globally trained chart-based parser as an alternative proposal distribution used in the approximate marginalization;
- propose effective methods for semisupervised learning, making the syntactic structure a latent variable;
- perform targeted syntactic evaluation and compare the model's performance with that of alternative models that are based on multitask learning.

Contents

Notation

$\mathbf{a}, \mathbf{b}, \ldots$	Vectors over the reals, *i.e.* $\mathbf{a} \in \mathbb{R}^m$.
$\mathbf{A}, \mathbf{B}, \ldots$	Matrices over the reals, *i.e.* $\mathbf{A} \in \mathbb{R}^{m \times n}$.
$[\mathbf{a}]_i$	Vector indexing: $[\mathbf{a}]_i \in \mathbb{R}$ for $1 \leq i \leq m$.
$[\mathbf{a}; \mathbf{b}]$	Vector concatenation: $\mathbf{a} \in \mathbb{R}^m$, $\mathbf{b} \in \mathbb{R}^n$, $[\mathbf{a}; \mathbf{b}] \in \mathbb{R}^{m+n}$.
$\mathbf{A} \circ \mathbf{B}$	Hadamard product $(\mathbf{A} \circ \mathbf{B})_{ij} = (\mathbf{A})_{ij}(\mathbf{B})_{ij}$.
$\mathcal{X}$	Finite vocabulary of words x.
$\mathcal{Y}(x)$	Finite set of trees y over a sentence x.
$\mathcal{V}(x)$	Finite set of labeled spans v over a sentence x.
$X, Y, \ldots$	Random variables with sample spaces $\mathcal{X}, \mathcal{Y}, \ldots$
x	A word from $\mathcal{X}$, outcome of random variable X.
y	A tree from $\mathcal{Y}(x)$, outcome of random variable Y.
x_1^m	A sequence of words $\langle x_1, \ldots, x_m \rangle$ from $\mathcal{X}^m$, shorthand: x.
$x_{<i}$	The sequence x_1^{i-1} preceding x_i.
P_X	Probability distribution.
p_X	Probability mass function.
$p(x)$	Probability $P(X = x)$.
p_θ, q_λ	Probability mass functions with emphasis on parameters.
$\mathbb{E}[g(X)]$	Expectation of $g(X)$ with respect to distribution P_X.
$H(X)$	Entropy of random variable X
Λ	Finite set of nonterminal labels.
$A, B, \ldots$	Nonterminal labels from Λ.
$S^\dagger$	Special root label not in Λ.
2^A	The poweset of set A.
$\mathbf{1}(p)$	Indicator function of predicate p.

1. Introduction

This thesis investigates the question: *What are effective ways of incorporating syntactic structure into a neural language model?*

We study a class of neural language models that explicitly model the hierarchical syntactic structure in addition to the sequence of words [Dyer et al., 2016, Buys and Blunsom, 2015b, 2018]. These models merges generative transition-based parsing with recurrent neural networks in order to model sentences together with their latent syntactic structure. The syntactic structure that decorates the words can be latent, and marginalized over, or can be given explicitly, for example as the prediction of an external parser. Although these are fundamentally joint model, they can be evaluated as regular language models (modeling only words) by (approximate) marginalization of the syntactic structure. In the case of the RNNG [Dyer et al., 2016], exact marginalization is intractable due to the parametrization of the statistical model, but importance sampling provides an effective approximate method. An externally trained discriminative parser is used to obtain proposal samples. Other models provide exact marginalization, but this typically comes at the cost of a less expressive parametrization, for example one in which the features cannot be structure-dependent [Buys and Blunsom, 2018].

In this thesis I study the RNNG [Dyer et al., 2016] and investigate:

The approximate marginalization I propose an alternative proposal distribution and investigate the impact.

- I propose a new discriminative chart-based neural parser that is trained with a global, Conditional Random Field (CRF), objective. The parser is an adaptation of the minimal neural parser proposed in Stern et al. [2017a], which is trained with a margin-based objective.
- This contrast with the choise of Dyer et al. [2016] for a transition-based parser as proposal, a discrminatively trained RNNG.
- We posit that a globally trained model is a better proposal distribution than a locally trained transition based model: a global model has ready access to competing analyses that can be structurally dissimilar but close in probability, whereas we hypothesize that a locally trained model is prone to produce locally corrupted structures that are nearby in transition-space. In a transition based parser more diverse samples can be obtained by flattening the transition distributions, which causes the model to be less confident in its predictions. The downside is that the model now explores parts of the probability space which it has not encountered during training.

Semi-supervised training by including unlabeled data To make joint models competitive language models they need to make use of the vast amounts of unlabeled data that exists.

- A major drawback of these syntactic language models is that they require annotated data to be trained, and preciously little of such data exists.
- We extend the training to the unsupervised domain by optimizing a variational lower bound on the marginal probabilities that jointly optimizes the parameters of proposal model ('posterior' in this framework) with the joint model.
- We obtain gradients for this objective using the score function estimator [Fu, 2006], also known as REINFORCE [Williams, 1992], which is widely used in the field of deep reinforcement learning, and we introduce an effective baseline based on argmax decoding [Rennie et al., 2017], which significantly reduces the variance in this optimization procedure.
- Our CRF parser particularly excels in the role of posterior thanks the independence assumptions that allow for efficient exact computation of key quantities: the entropy term in the lower bound can be computed exactly using Inside-Outside algorithm, removing one source of variance from the gradient estimation, and the argmax decoding can be performed exactly thanks to Viterbi, making the argmax baseline even more effective.

Alternative, simpler, models There are alternatives to the methods that this thesis investigates.

- Multitask learning of a neural language model with a syntactic side objective is a competitive and robust alternative method to infuse neural language models with syntactic knowledge.
- Training the syntactic model on data that mixes gold trees with predicted 'silver' trees for unlabeled data is a competitive and robust alternative to fully principled semi-supervised learning.
- We propose a simple multitask neural language model that predicts labeled spans from the RNN hidden states, using a feature function identical identical to that used in the CRF parser.
- We consider these alternatives in order to quantify significance of the latent structure and the semisupervised training as measured by some external performance metric.

Targeted syntactic evaluation TBA

2. Background

This section provides background on the four topics that are combined in this thesis: syntax, parsing, language modelling, and neural networks.

2.1. Syntax

We first introduce some concepts relating to syntax that are relevant for this thesis. In particular, we introduce the notion of a *constituent*, and the hierarchical organization of constituents in a sentence as described by phrase-structure grammars. The aim is to provide a succinct and compelling answer to the question: *Why should we care about constituency structure when modelling language?*

The exposition primarily follows Huddleston and Pullum [2002], a well established reference grammar of the English language that is relatively agnostic with respect to theoretical framework, with some excursions into Carnie [2010] and Everaert et al. [2015], which are less language-specific but rooted more in a particular theoretical framework[1].

We take the following three principles from Huddleston and Pullum [2002] as guiding:

1. *Sentences consist of parts that may themselves have parts.*
2. *These parts belong to a limited range of types.*
3. *The constituents have specific roles in the larger parts they belong to.*

To each principle we now dedicate a separate section.

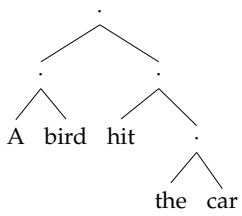
2.1.1. Constituents

Sentences consist of parts that may themselves have parts. The parts are groups of words that function as units and are called *constituents*. Consider the simple sentence *A bird hit the car.* The immediate constituents are *a bird* (the subject) and *hit the car* (the predicate). The phrase *hit the car* can be analyzed further as containing the constituent *the car*. The ultimate constituents of a sentence are the atomic words, and the entire analysis is called the constituent structure of the sentence. This structure can be indicated succinctly with the use of brackets

(1) [[A bird] [hit [the car]]]

or less succinctly as a tree diagram. Evidence for the existence of such constituents

[1] Broadly subsumable under the name *generative grammar*.



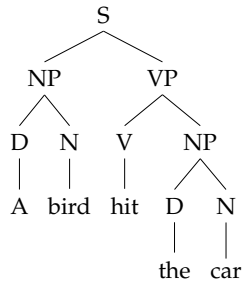
can be provided by examples such as the following, which are called constituent tests [Carnie, 2010]. Consider inserting the adverb *apparently* into our example sentence, to indicate the alleged status of the event described in the sentence. In principle there are six positions available for the placement of *apparently* (including before, and after the sentence). However, only three of these placements are actually permissible:[2]

(2) a. *Apparently* a bird hit the car.
b. *An *apparently* bird hit the car.
c. A bird *apparently* hit the car.
d. *A bird hit *apparently* the car.
e. *A bird hit the *apparently* car.
f. A bird hit the car, *apparently*.

Based on the bracketing that we proposed for this sentence we can formulate a general constraint: the adverb must not interrupt any constituent. Indeed, this would explain why *apparently* cannot be placed anywhere inside *hit the car* and not between *a* and *bird*. For full support, typically, results from many more such test are gathered, and in general these tests can be much more controversial than in our simple example [Carnie, 2010].

2.1.2. Categories

The constituents of a sentence belong to a limited range of types that form the set of syntactic categories. Two types of categories are distinguished: lexical and phrasal. The lexical categories are also known as part-of-speech tags. A tree can be represented in more detail by adding lexical (D, N, V) and phrasal categories (S, NP, VP). In this



[2]We use an asterisk * to indicate a sentence that is judged ungrammatical, as is customary in linguistics.

example, the noun (N) *car* is the head of the noun phrase (NP) *the car*, while the head of the larger phrase *hit the car* is the verb (V) *hit*, making this larger constituent a *verb phrase* (VP). The whole combined forms a sentence (S).

2.1.3. Hierarchy

The constituents have specific roles in the larger parts they belong to. This structure provides constraints that are not explainable from the linear order of the words themselves [Everaert et al., 2015]. Consider for instance the following example of the syntactic behaviour of *negative polarity items* (NPIs) from Everaert et al. [2015]. A negative polarity item is, to first approximation, a word or group of words that is restricted to negative context [Everaert et al., 2015].[3] Take the behaviour of the word *anybody*:

(3) a. The talk I gave did *not* appeal to *anybody*.
b. *The talk I gave appealed to *anybody*.
c. *The talk I did *not* give appealed to *anybody*.

From sentences (a) and (b) we might formulate the hypothesis that the word *not* must linearly precede the word *anybody*, but a counter example refutes this hypothesis: sentence (c) is also not grammatical. Instead—it is argued—the constraints that govern this particular pattern depend on hierarchical structure: the word *not* must 'structurally precede' the word *anybody* [Everaert et al., 2015]. Figure shows the constituent structure of both sentences. The explanation goes as follows: "In [the left tree] the hierarchical structure dominating *not* also immediately dominates the hierarchical structure containing *anybody*. In [the right tree], by contrast, *not* sequentially precedes *anybody*, but the triangle dominating *not* fails to also dominate the structure containing *anybody*." [Everaert et al., 2015].

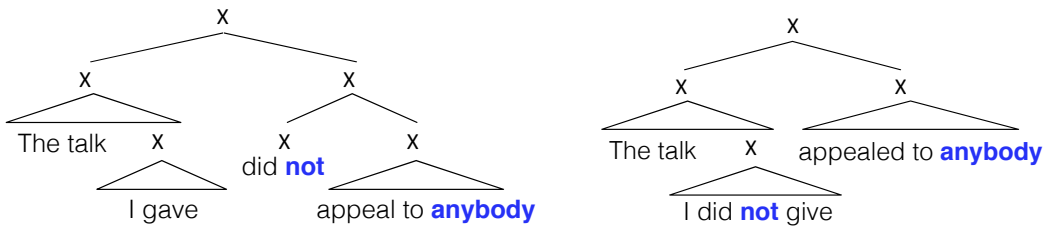


Figure 2.1.: Dependance on hierarchical structure of negative polarity items. Left shows the word *anybody* in the licensing context of *not*. Right shows the ungrammatical sentence where the word is not. Figure taken from Everaert et al. [2015]. The triangles indicate substructre that is not further explicated.

[3] More generaly, they are words that need to be licensed by a specific *licencing context* [Giannakidou, 2011].

2.1.4. Controversy

Theoretical syntax is rife with controversy, and wildly differing viewpoints exist. In fact, for each point made in our short discussion the exact opposite point has been made as well:

- Work in dependency grammar and other word-based grammar formalisms departs from the idea that lexical relations between individual words are more fundamental than constituents and their hierarchical organization [Tesnière, 1959, Nivre, 2005, Hudson, 2010], and dispenses with the notion of constituents altogether.
- A recurring cause of controversy is the question whether hierarchical structure needs to be invoked in linguistic explanation. That is, whether the kind of analysis we presented with embedded, hierarchically organized constituents is really fundamental to language. Frank et al. [2012] argue for instance that a shallow analysis of sentences into immediate constituents with linear order but no hierachical structure[4] is sufficient for syntactic theories, a claim that is vehemently rejected by Everaert et al. [2015].
- Research in cognitive neuroscience and psycholinguistics shows that human sentence processing is hierachical, giving evidence that processing crucially depends on the kind of structures introduced in the sections above [Hale, 2001, Levy, 2008, Brennan et al., 2016]. However, research also exists that shows that purely linear predictors are sufficient for modeling sentence comprehension, thus showing the exact opposite to be true [Conway and Pisoni, 2008, Gillespie and Pearlmutter, 2011, Christiansen et al., 2012, Gillespie and Pearlmutter, 2013, Frank et al., 2012].

Our work, however, takes a pragmatic position with respect to such questions: syntax, we assume, is whatever our dataset says it is. And to some degree, the question whether language is hierachical or linear is a question that this thesis engages with from a statistical and computational standpoint.

2.2. Parsing

Parsing is the task of predicting a tree y for a sentence x. Probabilitic parsers solve this problem by learning a probabilistic model p that describes a probability distribution over *all* the possible parses y of a sentence x. This distribution quantifies the uncertainty over the syntactic analyses of the sentence, and can be used to make predictions by finding the trees with highest probability. A further benefit of the probabilistic formulation is that the uncertainty over the parses can be determined quantitatively by computing the entropy of the distribution, and qualitatively by obtaining samples from the distribution. This section describes the form that such probabilistic models take, and the data from which they are estimated.

[4] A structure reminiscent of that predicted in the task of *chunking*, or shallow parsing.

2.2.1. Treebank

A treebank is a collection of sentences annotated with their grammatical structure that allows the estimation of statistical parsings models. The Penn Treebank [Marcus et al., 1993] is such a collection, consisting of sentences annotated with their constituency structure. Figure 2.2a shows an example tree from this dataset and figure 2.2b shows the tree after basic processing[5]. Some of the models in this work require trees to be in *normal form*: fully binary, but with unary branches at the terminals. Figure 2.2c show the result of this (invertible) normalization, that is obtained by introduction of an empty dummy label $\varnothing$. Annotating the labels with left and right endpoints of the words they span results in the tree in figure 2.2d.

A tree can be factorized into its parts: we can think of tree as a set of *labeled spans*, or as a set of *anchored rules*. A labeled span is a triple (A, i, j) of a syntactic label A from a labelset Λ together with the left and right endpoints i, j that the label spans. An *anchored rule* is a triple (r, i, j) or four-tuple (r, i, k, j), containing a rule r in normal form with span endpoints i, j, and a split-point k of the left and right child when r is not a lexical rule. Consider these two representations of the tree in figure 2.2d given in table 2.1. These different factorizations will become relevant in chapter 4 when formulating the probabilistic model.

Labeled spans	Anchored rules
(S, 0, 10)	(S $\rightarrow$ SBAR $\varnothing$, 0, 3, 10)
(SBAR, 0, 3)	(SBAR $\rightarrow$ WHNP S+VP, 0, 1, 3)
(WHNP, 0, 1)	(WHNP $\rightarrow$ *What*, 0, 1)
$\vdots$	$\vdots$
($\varnothing$, 9, 10)	($\varnothing \rightarrow$., 9, 10)

Table 2.1.: Two representations of the tree in 2.2d.

2.2.2. Models

The probabilistic model of a parser can be *discriminative*, describing the conditional probability distribution $p(y \mid x)$, or *generative*, describing the joint distribution $p(x, y)$. The form that the model takes is largely dictated by the algorithm used to build the parses.

Transition-based methods formulate parsing as a sequence of shift-reduce decisions made by a push-down automaton that incrementally builds a tree. The design of the transition system determines whether the tree is constructed bottom-up, top-down,[6] or

[5] This removes the functional tags and annotation of argument structure introduced in version 2 of the Penn Treebank [Marcus et al., 1994].

[6] Post-order and pre-order, respectively.

otherwise, and determines whether the trees need to be in normal form. The probabilistic model is defined over the sequences of actions, and the probability distribution typically factorizes as the product of conditional distributions over the next action given the previous actions. This makes it a directed model that is *locally normalized*.[7]

Definition 2.2.1. A probalistic model p over sequences $a \in \mathcal{A}^n$ is *locally normalized* if

$$\begin{aligned} p(a) &= \prod_{i=1}^{n} p(a_i \mid a_{<i}) \\ &= \prod_{i=1}^{n} \frac{\Psi(a_{<i}, a_i)}{Z(a_{<i})}, \end{aligned}$$

where $Z(a_{<i}) = \sum_{a \in \mathcal{A}} \Psi(a_{<i}, a)$ is a local normalizer and Ψ is a nonnegative scoring function: $\Psi(a_{<i}, a_i) \geq 0$ for all $a_{<i}$ and a_i.

Such a model is discriminative when the actions only build the tree, but can be made generative when word prediction is included in the action set.

Chart based parsing, on the other hand, factorizes trees along their parts, and defines the probabilistic model over the shared substructures. Representing the model as the product of local parts greatly reduces the number of variables required to specify the full distribution, and allows the use of dynamic programming for efficient inference. Discriminitive models based on Conditional Random Fields (CRF) [Lafferty et al., 2001] follow this factorization: local functions independently predict scores for the parts, and the product of this score is normalized *globally* by a normalizer that sums over the exponential number structures composable from the parts.

Definition 2.2.2. A probalistic model p over sequences $a \in \mathcal{A}^n$ is *globally normalized* if

$$p(a) = \frac{\Psi(a)}{Z},$$

where $Z = \sum_{a \in \mathcal{A}^n} \Psi(a)$ is a *global* normalization term, and $\Psi(a) \geq 0$ for all a. To allow efficient computation of the normalizer Z, the function Ψ typically factors over parts of a as $\Psi(a) = \prod_{i=1}^{K} \psi(a_i)$ with the choice of parts $a = \{a_i\}_{i=1}^{K}$ depending on the model.

Generative chart-based models, instead, estimate rule probabilities directly from a treebank. The probability of a tree is computed directly as the product of the probabilities of the rules that generate it, thus requiring no normalization.[8]

Either method have their advantages and disadvantages. The transition-based methods are fast, running in time linear in the sequence length, and allow arbitrary features that can condition on the entire sentence and the partially constructed derrivation. However, directed models with local normalization are known to suffer from the

[7] With the exception of those approaches that instead define a Conditional Random Field over the action sequences [Andor et al., 2016], in which case the model is defined over the globally normalized action sequences.

[8] In fact, this makes generative chart-based models directed, locally normalized models: they generate trees top-down by expanding localized normally rules.

problem of label bias [Lafferty et al., 2001], and conditioning on large parts of the partial derivations prohibits exact inference, so that decoding can be done only approximately, either with greedy decoding or with beam search. The chart based methods, on the other hand, allow efficient exact inference, and the global training objective that results from the global normalization receives information from all substructures. The method is much slower, however, running in time cubic in the sentence length for normal form trees [9] and linear in the size of the grammar[10], and the strong factorization of the structure, which makes the exact inference tractable, also means that features can condition only on small parts instead of large substructures.

A general challenge for sequential transition based models, related to the problem of label bias, is that their training is highly local: during training the models are only exposed to the individual paths that generate the example tree, which is a mere fraction of all the possible paths that the model is defined over.[11] Compare this with a globally normalized model, where the factorization over parts lets the model learn from an example tree about all the trees that share substructure.

In this thesis we investigate models where the scoring function Ψ, or its factorized version ψ, is implemented with neural networks. Chapter 3 describes a locally normalized transition-based model that has a discriminative and generative formulation, and chapter 4 introduces a globally normalized, discriminative chart-based parser.

2.2.3. Metrics

The standard evaluation for parsing is the Parseval metric [Black et al., 1991], which measures the number of correctly predicted labeled spans. The metric is defined as the harmonic mean, or F_1 measure, of the labelling recall R and precison P:

$$F_1 = \frac{2PR}{P+R}. \tag{2.1}$$

Let $\mathcal{R}$ be the set of labeled spans of the gold reference trees, and let $\mathcal{P}$ be the set of labeled spans of the predicted trees. The recall is the fraction of reference spans that were predicted

$$R = \frac{|\mathcal{R} \cap \mathcal{P}|}{|\mathcal{R}|},$$

[9] With greater exponents for trees that are not in normal form.

[10] Giving a total time complexity of $O(n^3|G|)$.

[11] One way to answer to this challenge is to use use a dynamic oracle during training [Goldberg and Nivre, 2013], also called exploration [Ballesteros et al., 2016, Stern et al., 2017a], which allow the model to explore paths that deviate from the single gold path in a principled way. The approach can be considered an instance of imitation learning [Vlachos, 2013, He et al., 2012]. In constituency parsing, dynamic oracles can produce substantial improvements in performance [Ballesteros et al., 2016], but they must be custom designed for each transition system [Fried and Klein, 2018]. However, we do not consider this directions in this thesis.

and the precision is the fraction of the predicted spans that is correct

$$P = \frac{|\mathcal{R} \cap \mathcal{P}|}{|\mathcal{P}|}.$$

The cannonical implementation of the Parseval metric is `EVALB` [Sekine and Collins, 1997].

2.3. Language models

A language model is a probability distribution over sequences of words. There are many ways to design such a model, and many datasets to estimate them from. This section focusses on those that are relevant to the models in this thesis.

2.3.1. Models

A language model is a probabilistic model p that assigns probabilities to sentences, or more formally, to sequences $x \in \mathcal{X}^*$ of any length over a finite vocabulary. Factorizing the probability of a single sequence $x \in \mathcal{X}^m$ over its timesteps gives the directed model:

$$p(x) = \prod_{i=1}^{m} p(x_i \mid x_{<i}). \tag{2.2}$$

This distribution can be approximated by lower order conditionals that condition on smaller, fixed-size, history, by making the Markov assumption assumption that

$$p(x_i \mid x_{<i}) = p(x_i \mid x_{i-j-1}^{i-1}).$$

This is the approach taken by n-gram language models. The lower order conditionals can be estimated directly by smoothing occurence counts [Chen and Goodman, 1999, Kneser and Ney, 1995], or they can be estimated by locally normalized scores $\Psi(x_{i-j-1}^{i-1}, x_i)$, given by a parametrized function Ψ. This function can be log-linear or a non-linear neural network [Rosenfeld, 1996, Bengio et al., 2003]. The lower order approximation can also be dispensed with, making the model more expressive, but the estimation problem much harder. Language models based on recurrent neural networks (RNNs) follow this approach by using functions that compute scores $\Psi(x_{<i}, x_i)$ based on the entire history [Mikolov et al., 2010]. These models, and in particular the approaches based on the LSTM variant of the RNN [Hochreiter and Schmidhuber, 1997], have shown to be remarkably effective and substantially improve over the above methods to give the current state of the art [Zaremba et al., 2014, Jozefowicz et al., 2016, Melis et al., 2017].[12] Other neural network architectures based on convolutions [Kalchbren-

[12] Although recent work shows that the effective memory depth of RNN models is much smaller than the unbounded history suggests: Chelba et al. [2017] show that, in the perplexity they assign to test data, an RNN with unbounded history can be approximated very well by an RNN with bounded history. To be precise: a neural n-gram language model, with the fixed-size history encoded by a bidirectional RNN, is equivalent to an RNN with unbounded history for $n = 13$, and to an RNN that is reset at the start of sentences for $n = 9$. This finding is consistent accross dataset sizes.

ner et al., 2014] and stacked feedforward networks with 'self-attention' [Vaswani et al., 2017] have been succesfully applied to language modelling with unbounded histories as well.

Alternatively, a language model can be obtained by marginalizing a structured latent variable h in a joint model $p(x, h)$:

$$p(x) = \sum_{h \in \mathcal{H}} p(x, h). \tag{2.3}$$

The structure of h allows this joint distribution to be factorized in a ways very much unlike that of equation 2.2. Such language models are defined for example by a Probabilistic Context Free Grammar (PCFG), in which case h is a tree, and a Hidden Markov Model (HMM), in which case h is a sequence of tags. The strong independence assumptions of these models allows the marginalization to be computed efficiently, but also disallows the models to capture arbitrary dependencies in x, which is precisely what we want from a language model. The recurrent neural network grammar (RNNG) [Dyer et al., 2016] model introduced in chapter 3 also defines a language model through a joint distribution, but that model is factorized as a sequential generation process over both the structure h and the sequence x, which makes it a competitive language model, but at the price of losing efficient exact marginalization.

This thesis focusses on language models that incorporate syntax. The models have precedents. Most directly related to our discussion are: language models obtained from top-down parsing with a PCFG [Roark, 2001]; syntactic extensions of n-gram models with count-based and neural network estimation [Chelba and Jelinek, 2000, Emami and Jelinek, 2005]; and a method that is reminiscent of n-gram methods, but based on arbitrary overlapping tree fragments [Pauls and Klein, 2012].

2.3.2. Data

Language models enjoy the benefit that they require no labeled data; any collection of tokenized text can be used for training. In this work we focus on English language datasets. The sentences in the Penn Treebank have long been a popular dataset for this task. More recently has seen the introduction of datasets of much greater size, such as the One Billion Word Benchmark [Chelba et al., 2013] that consists of news articles, and datasets that focus on long-distance dependencies, such as the Wikitext datasets [Merity et al., 2016] that consists of Wikipedia articles grouped into paragraphs.

2.3.3. Metrics

The standard metric to evaluate a language model is the *perplexity* per token that it assigns to held out data. The lower the perplexity, the better the model. Perplexity is an information theoretic metric that corresponds to an exponentiated estimate of the model's entropy normalized over number of predictions, measured in nats, and was introduced for this purpose by Jelinek [1997][13]. The metric can be interpreted as the

[13] According to [Chelba et al., 2017].

average number of guesses needed by the model to predict each word from its left context.

Definition 2.3.1. The *perplexity* of a language model p on a sentence x of length m is defined as

$$\exp\left\{-\frac{1}{m}\log p(x)\right\}.$$

The perplexity on a set of sentences $\{x^{(1)}, \ldots, x^{(N)}\}$ is defined as the exponentiated mean over all words together:

$$\exp\left\{-\frac{1}{M}\sum_{n=1}^{N}\log p(x^{(n)})\right\},$$

where $M = \sum_{n=1}^{N} m_n$ is the sum of all the sentence lengths m_n.

The appeal of perplexity is that it is an aggregate metric, conflating different causes of succes when predicting the next word. This conflation is also its main shortcoming, making it hard to determine whether the model has robustely learned high-level linguistic patterns such as those described in syntactic and semantic theories. For this reason, alternative methods have been proposed to evaluate language models: evaluation with adversarial examples [Smith, 2012]; prediction of long distance subject-verb agreement [Linzen et al., 2016]; and eliciting syntactic acceptability judgments [Marvin and Linzen, 2018]. Chapter 5 discusses these alternatives in greater detail, and demonstrates evalution with the method proposed in [Marvin and Linzen, 2018].

2.4. Neural networks

In this thesis we use neural networks to parametrize probability distributions. We consider a neural network an abstraction that denotes a certain type of parametrized differentiable function. Let x be a word from a finite vocabulary $\mathcal{X}$, and let $\mathbf{x}$ and $\mathbf{y}$ be vectors in respectively $\mathbb{R}^n$ and $\mathbb{R}^m$.

Definition 2.4.1. A *word embedding* is vector representation of a word, assigned by an embeding function E that takes elements from $\mathcal{X}$ to $\mathbb{R}^n$:

$$\mathbf{x} = \mathrm{E}(x).$$

The function can be a simple lookup table, or a more elaborate function that depends for example on the orthography of the word.

Definition 2.4.2. A *feedforward neural network* is a parametrized function FFN from $\mathbb{R}^n$ to $\mathbb{R}^m$:

$$\mathbf{y} = \textsc{Ffn}(\mathbf{x}).$$

Internally, the function computes stacks of affine transformations followed by an elementwise application of a nonlinear function. The number of repetitions of these applications is refered to as the number of layers of the network.

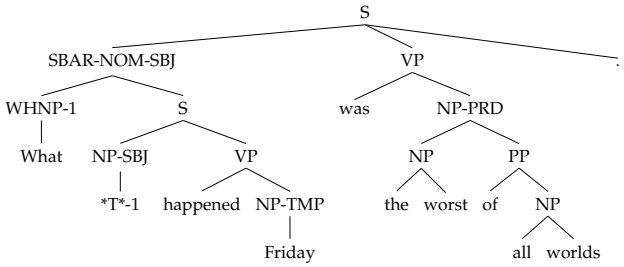
Definition 2.4.3. A *recurrent neural network* (RNN) is a parametrized function RNN that takes a sequence of vectors $\mathbf{x}_1^k = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k]$, each in $\mathbb{R}^n$, and produces a sequence of output vectors $[\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n]$ each in $\mathbb{R}^m$:

$$[\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_k] = \text{RNN}(\mathbf{x}_1^k).$$

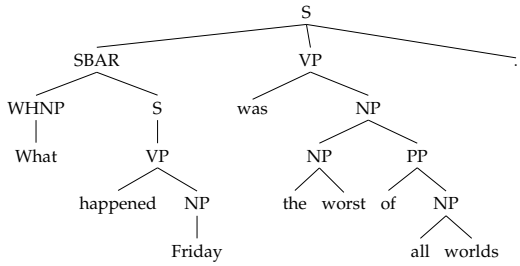
Each vector $\mathbf{y}_i$ is a function of the vectors $[\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_i]$, for which reason we refer to the vector $\mathbf{y}_i$ as a context-dependent *encoding* of the vector $\mathbf{x}_i$. An RNN can be applied to the input sequence in reverse. This makes each $\mathbf{y}_i$ a function of the vectors $[\mathbf{x}_i, \mathbf{x}_{i+1}, \ldots, \mathbf{x}_k]$. We denote the output of the forward direction with $\mathbf{f}$ and the output of the backward direction with $\mathbf{b}$.

Definition 2.4.4. An RNN is *bidirectional* when it combines the output of an RNN that runs in the forward direction, with the output of an RNN that runs in the backward direction. Combining their output vectors by concatenation gives for each position a vector $\mathbf{h}_i = [\mathbf{f}_i; \mathbf{b}_i]$ that is a function of the entire input sequence.

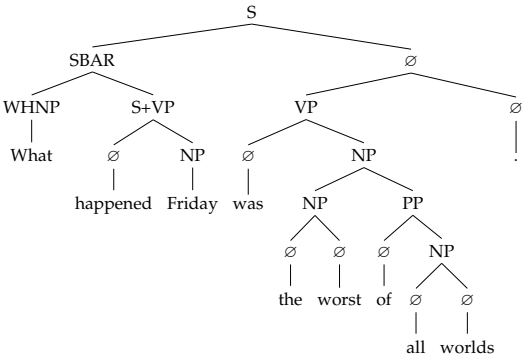
Definition 2.4.5. An LSTM [Hochreiter and Schmidhuber, 1997] is a particular way to implement the internals of the RNN function. It is the only type of RNN used in this work, and we use the two names exchangeably.



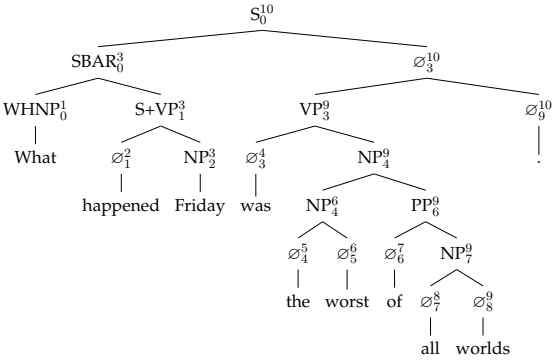
(a) Original Penn Treebank tree.



(b) Function tags and traces removed.



(c) Converted to normal form using a dummy label ∅.



(d) In normal form with spans.

Figure 2.2.: Converting a treebank tree (withouth part-of-speech tags).

3. Recurrent Neural Network Grammars

This chapter describes the Recurrent Neural Network Grammar (RNNG), a probabilistic model of sentences with explicit phrase structure introduced by Dyer et al. [2016]. The model has a discriminative and generative formulation, but both are based on a shift-reduce transition system that builds trees in top-down order and define locally normalized probability distributions over action sequences. While the discriminative model is a constituency parser, the generative model is a joint model that can be evaluated both as constituency parser and as language model. The strength of both models lies in their unbouded history: at each step the model can condition on the entire derivation constructed thus far. This derivation is summarized by a syntax-dependent recurrent neural networks. This parametrization without independence assumptions, however, does preclude the use of exact inference—for example based on dynamic programming—but approximate inference based on greedy decoding and importance sampling provides tractable alternatives.

In this chapter:

- We describe the discriminative and generative formulations of the RNNG in detail.
- We describe the approximate inference algorithms that allow the RNNG to be used as parser and as language model.
- We implement the models and reproduce the results of Dyer et al. [2016].

3.1. Model

The discriminative RNNG is a transition based parser that uses regular RNNs to summarize the actions in the history and the words on the buffer into vectors, and uses a special RNN with a syntax-dependent recurrence to obtain a vector representation of items on the the stack. The generative RNNG can be seen as a generative formulation of the RNNG that jointly models the words instead of merely conditioning on them. Alternatively, it can be seen as a structured model of language that predicts words together with their structure. From the parsing perspective, the generative RNNG simply dispenses with the buffer to instead predict the words. But as a model of sentences, it can best be understood as kind of structured RNN: it predicts words incrementally, but compresses and labels them recursively whenever they form a complete constituents. In this section we introduce both models, starting at their backbone: the transition system.

3.1.1. Transition sytem

Both RNNG models use a transition system crafted for top-down parsing. The discriminative transition system has three actions:

- OPEN(X) opens a new nonterminal symbol X in the partially constructed tree.
- SHIFT moves the topmost word x from the buffer onto the stack.
- REDUCE closes the open constituent by composing the symbols in it into a single item representing the content of the subtree. This allows nodes with an arbitrary number of children.

How these actions work to build a tree is best illustrated with an example derivation.

Example 3.1.1. (Discriminative transition system) The following table shows the consecutive states of the parser that produces the gold tree for input sentence *The hungry cat sleeps*. Individual items are separated by the midline symbol.

	Stack	Buffer	Action
0		*The* | *hungry* | *cat* | *meows* | .	OPEN(S)
1	(S	*The* | *hungry* | *cat* | *meows* | .	OPEN(NP)
2	(S | (NP	*The* | *hungry* | *cat* | *meows* | .	SHIFT
3	(S | (NP | *The*	*hungry* | *cat* | *meows* | .	SHIFT
4	(S | (NP | *The* | *hungry*	*cat* | *meows* | .	SHIFT
5	(S | (NP | *The* | *hungry* | *cat*	*meows* | .	REDUCE
6	(S | (NP *The hungry cat*)	*meows* | .	OPEN(VP)
7	(S | (NP *The hungry cat*) | (VP	*meows* | .	SHIFT
8	(S | (NP *The hungry cat*) | (VP | *meows*	.	REDUCE
9	(S | (NP *The hungry cat*) | (VP *meows*)	.	SHIFT
10	(S | (NP *The hungry cat*) | (VP *meows*) | .		REDUCE
11	(S (NP *The hungry cat*) (VP *meows*) .)		

Table 3.1.: Discriminative transition system. Example from Dyer et al. [2016].

The actions are constrained in order to only derive well-formed trees:

- SHIFT can only be excecuted if there is at least one open nonterminal (all words must be under some nonterminal symbol).
- OPEN(X) requires there to be words left on the buffer (all constituents must eventually hold terminals).[1]
- REDUCE requires there to be at least one terminal following the open nonterminal, and the topmost nonterminal can only be closed when the buffer is empty (all nodes must end under a single root node).

[1] Additionally, the number of open nonterminals can be at most some arbitrary number n, in practice 100. This constraint prevents the model from generating trees with arbitrarily long unary chains.

The generative transition system is derived from this system by replacing the SHIFT action, which moves the word x from the buffer into the open constituent on the stack, with an action that *predicts* the word:

- GEN(x) predicts that x is the next word in the currently open constituent, and puts this word on the top of the stack.

The buffer is dispensed with and is replaced by a similar structure that records the sequence words predicted so far.

Example 3.1.2. (Generative transition system) Generating the sentence of example 3.1.1 together with its tree.

	Stack	Terminals	Action
0			OPEN(S)
1	(S		OPEN(NP)
2	(S | (NP		GEN(*The*)
3	(S | (NP | *The*	*The*	GEN(*hungry*)
4	(S | (NP | *The* | *hungry*	*The* | *hungry*	GEN(*cat*)
5	(S | (NP | *The* | *hungry* | *cat*	*The* | *hungry* | *cat*	REDUCE
6	(S | (NP *The hungry cat*)	*The* | *hungry* | *cat*	OPEN(VP)
7	(S | (NP *The hungry cat*) | (VP	*The* | *hungry* | *cat*	GEN(*meows*)
8	(S | (NP *The hungry cat*) | (VP | *meows*	*The* | *hungry* | *cat* | *meows*	REDUCE
9	(S | (NP *The hungry cat*) | (VP *meows*)	*The* | *hungry* | *cat* | *meows*	GEN(.)
10	(S | (NP *The hungry cat*) | (VP *meows*) | .	*The* | *hungry* | *cat* | *meows* | .	REDUCE
11	(S (NP *The hungry cat*) (VP *meows*) .)	*The* | *hungry* | *cat* | *meows* | .	

Table 3.2.: Generative transition system. Example from Dyer et al. [2016].

With the transition system in place, we now move to describe how the RNNG defines a probability distribution over the sequeunces of transition actions.

3.1.2. Model

Fundamentally, the model is a probability distribution over transition action sequences $a = \langle a_1, \ldots, a_T \rangle$ that generate trees y. Conditionally—given a sequence of words x—in the discriminative model, and jointly—predicting the sequence x—in the generative model. The model then defines a distribution over trees $\mathcal{Y}(x)$ through the bijective transformation that maps transition sequences a to trees y and vice versa.[2] Put simply, the model is thus defined as

$$p(a) = \prod_{t=1}^{T} p(a_t \mid a_{<t}), \tag{3.1}$$

where in the discriminative case $p(a) = p(y \mid x)$ and in the generative model $p(a) = p(x, y)$. The exact model however is slightly more complicated, a consequence of the

[2] Note that as a result of the transition system, the trees in the set $\mathcal{Y}(x)$ are rather unconstrained: the trees can have any arity, and can contain arbitrarily long unary chains, depending on the maximum number of open nonterminals n chosen.

difference between the discriminative and the generative actions and of practical concerns regarding the implementation. We will define the model more precisely. For this we need to introduce some things.

First we define the set of discriminative actions as

$$\mathcal{A}_D = \{\text{SHIFT}, \text{OPEN}, \text{REDUCE}\}, \tag{3.2}$$

and the set of generative actions as

$$\mathcal{A}_G = \{\text{GEN}, \text{OPEN}, \text{REDUCE}\}. \tag{3.3}$$

The finite set of nonterminals is denoted by Λ and the finite alphabet of words by $\mathcal{X}$. For the discriminative model a is sequence over $\mathcal{A}_D$, and in the generative model a is a sequence over $\mathcal{A}_G$. In both cases a is restricted to sequences that form a valid tree y. The sequence of nonterminals $n = \langle n_1, \ldots, n_K \rangle$ from Λ^K is the sequence of nonterminal nodes obtained from a tree y by pre-order traversal, and we let a sentence x be an element of $\mathcal{X}^N$. Finally, we introduce two functions that map between sets of indices to indicate the number of times a particular action has been taken at each time step:

$$\mu_a : \{1, \ldots, T\} \to \{1, \ldots, K\} : t \mapsto \sum_{i=1}^{t-1} \mathbf{1}(a_i = \text{OPEN}),$$

and

$$\nu_a : \{1, \ldots, T\} \to \{1, \ldots, N\} : t \mapsto \sum_{i=1}^{t-1} \mathbf{1}(a_i = \text{GEN}),$$

but for brevity we drop the subscript a. We are now in the position to write down the exact models.

Definition 3.1.3. (Discriminative RNNG) Let a be a sequence over $\mathcal{A}_D$ of length T. Then the model for the discrminative RNNG is

$$p(y \mid x) = p(a \mid x) = \prod_{t=1}^{T} P(a_t \mid x, a_{<t}),$$

where

$$P(a_t \mid x, a_{<t}) = \begin{cases} p(a_t \mid x, a_{<t}) p(n_{\mu(t)} \mid x, a_{<t}) & \text{if } a_t = \text{OPEN}, \\ p(a_t \mid x, a_{<t}) & \text{otherwise.} \end{cases}$$

Whenever the prediction is to open a new nonterminal the action probability factorizes as the product of two separate probabilities: the probability of the action itself, and the probability of the next upcoming label in n, which is indexed by $\mu(t)$.

Definition 3.1.4. (Generative RNNG) Let a be a sequence over $\mathcal{A}_G$ of length T, which include the actions that generate words.[3] The model for the generative RNNG then is

$$p(y \mid x) = p(a) = \prod_{t=1}^{T} P(a_t \mid a_{<t}),$$

where

$$P(a_t \mid a_{<t}) = \begin{cases} p(a_t \mid a_{<t})p(n_{\mu(t)} \mid a_{<t}) & \text{if } a_t = \text{OPEN}, \\ p(a_t \mid a_{<t})p(x_{\nu(t)} \mid a_{<t}) & \text{if } a_t = \text{GEN}, \\ p(a_t \mid a_{<t}) & \text{otherwise.} \end{cases}$$

This corresponds with the discriminative model on the actions that they share. For the action that predicts the next word, the factorization is identical to that of the action that predicts the next nonterminal, with the difference that the next word in x is instead indexed by $\nu(t)$.

The probabilities over next actions at time step t are given by classifiers on a vector $\mathbf{u}_t$,[4] which represents the parser configuration at that timestep:[5]

$$p(a_t \mid a_{<t}) \propto \exp\left\{[\text{FFN}_\alpha(\mathbf{u}_t)]_{a_t}\right\} \tag{3.4}$$

$$p(n_{\mu(t)} \mid a_{<t}) \propto \exp\left\{[\text{FFN}_\beta(\mathbf{u}_t)]_{n_{\mu(t)}}\right\} \tag{3.5}$$

$$p(x_{\nu(t)} \mid a_{<t}) \propto \exp\left\{[\text{FFN}_\gamma(\mathbf{u}_t)]_{x_{\nu(t)}}\right\} \tag{3.6}$$

The parameters α, β, and γ are separate sets of parameters. The computation of the vector $\mathbf{u}_t$ is described in the next section.

Remark 3.1.5. We could have defined the actions as

$$\mathcal{A}_D = \{\text{REDUCE}, \text{SHIFT}\} \cup \{\text{OPEN}(n) \mid n \in \Lambda\},$$

and

$$\mathcal{A}_G = \{\text{REDUCE}\} \cup \{\text{OPEN}(n) \mid n \in \Lambda\} \cup \{\text{GEN}(x) \mid x \in \mathcal{X}\},$$

and defined $p(a)$ as in 3.1. However, in the case of the generative model this is particularly inefficient from a computational perspective. Note that the set $\mathcal{X}$ is generally very large[6], and the normalization in 3.6 requires a sum over all actions while a large number of the actions do not generate words. Besides, the presentation in definitions 3.1.3 and 3.1.4 is conceptually cleaner: first choose an action, then, if required, choose the details

[3] Note that under this action set $p(a_t \mid a_{<t}, x_{<\nu(t)}) = p(a_t \mid a_{<t})$, given the fact that the words in $x_{<\nu(t)}$ are contained in $a_{<t}$.

[4] For brevity we omit the conditioning on x, which was redundant already in the case of the generative model.

[5] Where we pretend that a_t, n_t, and x_t double as indices.

[6] On the order of tens or hundreds of thousands.

of that action. For these reasons we opt for the two-step prediction. For consistency we extend this modelling choice to the discriminative RNNG. And although it appears that Dyer et al. [2016] model the sequences according to 3.1, followup work takes our approach and models the actions of the generative RNNG as in definition 3.1.4 [Hale et al., 2018].

3.2. Parametrization

The transition probabilities are computed from the vector $\mathbf{u}_t$ that summarizes the parser's entire configuration history at time t. This vector is computed incrementally and in a syntax-dependent way. It is defined as the concatenation of three vectors, each summarizing one of the three datastructures separately:

$$\mathbf{u}_t = [\mathbf{s}_t; \mathbf{b}_t; \mathbf{h}_t].$$

Here, $\mathbf{s}_t$ represents the stack, $\mathbf{b}_t$ represents the buffer, and $\mathbf{h}_t$ represents the history of actions.

The vectors $\mathbf{b}_t$ and $\mathbf{h}_t$ are computed each with a regular RNN: the history vector is computed in the forward direction, and the buffer is encoded in the backward direction to provide a lookahead. The vector $\mathbf{s}_t$ represents the partially constructed tree that is on the stack, and its computation depends on this partial structure by using a structured RNN that encodes the tree in top-down order while recursively compressing constituents whenever they are completed.

3.2.1. Stack encoder

The stack RNN computes a representation based on incoming nonterminal and terminal symbols while these are respectively opened and shifted, as a regular RNN would, but rewrites this history whenever the constituent that they form is closed. Whenever a REDUCE action is predicted, the RNN rewinds its hidden state to before the constituent was opened; the items making up the constinuent are composed into a single vector by a composition function; and this composed vector is then fed into the rewound RNN as if the subtree were a single input. The closed constituent is now a single item represented by a single vector. Due to the nested nature of constituents this procedure recursively compresseses subtrees.

Example 3.2.1. (Composition function) Consider the state of the parser in example 3.1.1 at step 5, when the stack contains the five items

$$(\text{S} \mid (\text{NP} \mid \textit{The} \mid \textit{hungry} \mid \textit{cat}.$$

Each first represented by an embedding vector, and then encoded by an RNN in the regular way. A REDUCE action is now predicted: the items are popped from the stack until the first open bracket is popped, which in this example is the item that opens the

NP bracket. This process also rewinds the RNN state to before the bracket was openend, which in this example brings the RNN back to its state at step 1. The composition now computes a vector representation for the four popped items, returning a single representation for the composed item

(NP *The hungry cat*).

The RNN is fed this reduced input, resulting in the encoding of stack at step 6, and finalizing the reduction step. This process is recursive: consider the reduction that takes the stack from the five items

(S | (NP | *The* | (ADJP *very hungry*) | *cat*

to the two items

(S | (NP *The* (ADJP *very hungry*) *cat*).

The items in the constituent (ADJP *very hungry*) have already been composed into a single item, and now it takes part in the composition at a higher level.

3.2.2. Composition function

Two kinds of functions have been proposed to for the composition described above: the original function based on a bidirectional RNN [Dyer et al., 2016], and a more elaborate one that additionally incorporates an attention mechanism [Kuncoro et al., 2017]. Both methods encode the invidual items that make up the constituent with a bidirectional RNN, but where the simpler version merely concatenates the endpoint vectors, the attention based method computes a convex combination of all these vectors, weighted by predicted attention weights. Kuncoro et al. [2017] show that the models with the attention-based composition outperform the models with the simpler composition, and that the attention mechanism adds interpretability to the composition. For these reasons we only consider the attention-based composition.

The attention-based composition is defined as follows. Let $\mathbf{h}_1, \ldots, \mathbf{h}_m$ be the vector representations computed by a dedicated bidirectional LSTM for the m words making up the constituent that is to be compressed, and let $\mathbf{n}$ be the learned embedding for the nonterminal category of that constituent. The attention weight a_i for the i-th position is computed as the exponentiated bilinear product

$$a_i \propto \exp\left\{\mathbf{h}_i^\top \mathbf{V}\,[\mathbf{u}_t; \mathbf{n}]\right\}$$

between the vector representation $\mathbf{h}_i$ and the concatenation of current parser state representation $\mathbf{u}_t$ and nonterminal representation $\mathbf{n}$. The values $a_1, \ldots, a_m$ are normalized to sum to 1, and the matrix $\mathbf{V}$ is part of the trainable parameters. The convex combination of the vectors $\mathbf{h}_i$ weighted by values a_i gives the vector

$$\mathbf{m} = \sum_{i=1}^{m} a_i \mathbf{h}_i,$$

which represents the words in the constituent, and the final representation of the constituent is the gated sum

$$\mathbf{c} = \mathbf{g} \circ \mathbf{n} + (1 - \mathbf{g}) \circ \mathbf{m}.$$

of this vector and the vector $\mathbf{n}$ representing the nonterminal, weighted by a gating vector $\mathbf{g}$. The vector of gates $\mathbf{g}$ is computed by an affine transformation on $[\mathbf{n}; \mathbf{m}]$ followed by an elementwise application of the logistic function.[7] This final representation can weigh the contribution to the final representation of the words and their nonterminal category, depending on the context.

3.3. Training

The discriminative and generative model are trained to maximize the objective

$$\mathcal{L}(\theta) = \sum_{(x,y)\in\mathcal{D}} \log p_\theta(y \mid x),$$

respectively

$$\mathcal{L}(\theta) = \sum_{(x,y)\in\mathcal{D}} \log p_\theta(x, y),$$

by gradient-based optimization on the parameters θ. We perfom iterative parameter updates, optaining new parameters $\theta^{(t+1)}$ at timestept $t + 1$ from parameters $\theta^{(t)}$ at timestep t using the update rule

$$\theta^{(t+1)} = \theta^{(t)} + \gamma^{(t)} \nabla_{\theta^{(t)}} \mathcal{L}(\theta^{(t)}),$$

where the value $\gamma^{(t)}$ is a time-indexed learning rate. We replace $\nabla_\theta \mathcal{L}(\theta)$ with an unbiased estimate

$$\sum_{k=1}^{K} \nabla_\theta \log p_\theta(x^{(k)}, y^{(k)}),$$

where pairs $(x^{(k)}, y^{(k)})$ are sampled uniformly from $\mathcal{D}$, which works because for uniformly random sampled pairs $(x^{(s)}, y^{(s)})$ from the dataset.

$$\nabla_\theta \mathcal{L}(\theta) \propto \mathbb{E}[\nabla_\theta \log p_\theta(x^{(s)}, y^{(s)})].$$

We use minibatches for this estimates, making our optimization procedure stochastic. Under certain conditions on $\gamma^{(t)}$ this method is guaranteed to converge to a local optimum [Robbins and Monro, 1951].

[7] Our definition of $\mathbf{c}$ is slightly different from that of Kuncoro et al. [2017]: where they use another vector representation $\mathbf{t}$ different from $\mathbf{n}$ in the above definition; we use just n to compute both the attention weights and the final vector.

3.4. Inference

The two formulations of the RNNG have complementary applications: both models can be used for parsing, but the generative model can additionally be used as a language model. How these problems can be solved is the topic of this section.

3.4.1. Discriminative model

Parsing a sentence x with the discriminative model corresponds to solving the following search problem of finding the maximum *a posteriori* (MAP) tree

$$\hat{y} = \arg\max_{y \in \mathcal{Y}(x)} p_\theta(y \mid x).$$

Solving this exactly is intractable due to the parametrization of the model. At each timestep, the model conditions on the entire derivation history which excludes the use of dynamic programming to solve this efficiently. Instead we rely on an approximate search strategy. There are two common approaches for this: greedy decoding and beam search. We only focus on the first. Greedy decoding is precisely what it suggests: at each timestep we greedily select the best local decision

$$\hat{a}_t = \arg\max_{a} p_\theta(a \mid \hat{a}_{<t}).$$

The predicted parse is then the approximate MAP tree $y^* = \text{yield}(\hat{a})$, which is the tree constructed from the sequence $\hat{a} = \langle \hat{a}_1, \ldots, \hat{a}_m \rangle$.

3.4.2. Generative model

Given a trained generative model p_θ we are interested in solving the following two problems: parsing a sentence x

$$\hat{y} = \arg\max_{y \in \mathcal{Y}(x)} p_\theta(x, y),$$

and computing its marginal probability

$$p(x) = \sum_{y \in \mathcal{Y}(x)} p_\theta(x, y).$$

Either inference problem is intractable as either problem would require exhaustive enumeration of and computation over all possible action sequences. Luckily, both can be effectively approximated with the same method: importance sampling using a conditional proposal distribution $q_\lambda(y \mid x)$ [Dyer et al., 2016].

Approximate marginalization The proposal distribution allows us to rewrite the marginal probability of a sentence as an expectation under this distribution:

$$\begin{aligned} p(x) &= \sum_{y \in \mathcal{Y}(x)} p_\theta(x, y) \\ &= \sum_{y \in \mathcal{Y}(x)} q_\lambda(y \mid x) \frac{p_\theta(x, y)}{q_\lambda(y \mid x)} \\ &= \mathbb{E}_q \left[\frac{p_\theta(x, y)}{q_\lambda(y \mid x)} \right] \end{aligned}$$

This expectation can be approximated with a Monte Carlo estimate

$$\mathbb{E}_q \left[\frac{p_\theta(x, y)}{q_\lambda(y \mid x)} \right] \approx \frac{1}{K} \sum_{k=1}^{K} \frac{p_\theta(x, y^{(k)})}{q_\lambda(y^{(k)} \mid x)}, \tag{3.7}$$

using proposal samples $y^{(k)}$ sampled from the proposal model q_λ conditioning on x.

Approximate MAP tree To approximate the MAP tree $\hat{y}$ we use the same set of proposal samples as above, and choose the tree y from the proposal samples that has the highest probability probability under the joint model $p_\theta(y, x)$.

Proposal distribution The proposal distribution must have a support that covers that of the joint model. This corresponds to the property that for all x and $y \in \mathcal{Y}(x)$

$$p(x, y) > 0 \Rightarrow q(y \mid x) > 0.$$

We additionally want that samples can be obtained efficiently, and that their conditional probability can be computed. All these requirements are met by the discriminative RNNG: samples can be obtained by ancestral sampling over the transition sequences. For this reason Dyer et al. [2016] use this as their proposal. However, any discriminatively trained parser that meets these requirements can be used alternatively, and in chapter 4 we will propose such an alternative.

3.5. Experiments

This section reports the results of the experiments performed with our own implementation of the RNNG. We train the discriminative and generative model on the Penn Treebank with the same hyperparameters as Dyer et al. [2016], and compare our results to conclude the correctness of our implementation. We then investigate how the approximate marginalization depends on two parameters: the number of samples, and the temperature used to scale the distribution of the proposal model. For the general training setup and details about data and optimization we refer the reader to appendix A.

3.5.1. Setup

We use the same hyperparameters as [Dyer et al., 2016]. For both models the embeddings are of dimension 100, and the LSTMs have 2 layers. The LSTMs have dimension 128 in the discriminative model and 256 in the generative model, and the dimension of the feedforward networks that compute the action distributions are the same as the LSTMs. We use weight decay of 10^{-6}, and apply dropout to all layers including the recurrent connections[8] using a dropout ratio of 0.2 in the discriminative model and 0.3 in the generative model. Given the above dimensions, the total number of trainable parameters is approximately 0.8M for the discriminative model and 9.6M for the generative model (the exact numbers are given in table A.1).

Our discriminative model does not use tags, which contrasts with Dyer et al. [2016] who use tags predicted by an external tagger. This decision was made with the application of the RNNG as language model in mind, in which case tags are not available, but this choice will affect the parsing accuracy of the discriminative model. For a more elaborate justification of this choice we refer the reader to section of the implementation appendix.

3.5.2. Results

We train 10 separate models from different random seeds.[9] We report mean and standard deviation to give a sense of the variability of the accuracy, as well as the value of the model with the best development performance. Model selection is based on development F1 for the discriminative model, and based on development perplexity for the generative model. For inference with the generative model we follow Dyer et al. [2016] and sample 100 trees from our best discriminative model. The samples are obtained by ancestral sampling on the action sequeunces, and we flatten this ditribution by raising each action distribution to the power $\alpha = 0.8$ and renormalizing it.

The parsing F1 of the discriminative model are shown in table 4.1, and the results of the generative model are shown in tables 3.4 (F1) and 3.5 (perplexity). The value between brackets is the value based on model selection.

	Ours	Dyer et al. [2016]
F1	$88.47 \pm 0.17 \quad (88.58)$	– (91.7)

Table 3.3.: F1 scores for the discriminative RNNG.

The accuracy of our discriminative model is below that of Dyer et al. [2016], a gap that we think can reasonably be attributed to the absence of tag information in our implementation. This gap in F1 is carried through in a lesser degree to the generative model, resulting in a lower F1 than reported by Dyer et al. [2016] when using our own

[8] Also called *variational dropout* [Gal and Ghahramani, 2016], which is implemented by default in Dynet [Neubig et al., 2017a].

[9] The random seeds control both the random initialization of the model parameters as well as the order in which we pass through the training data.

	Our samples	Dyer et al. [2016]
Our RNNG	$91.07 \pm 0.1 \quad (91.12)$	$93.32 \pm 0.1 \quad (93.32)$
Dyer et al. [2016]	–	– (93.3)
Kuncoro et al. [2017]	–	– (93.5)

Table 3.4.: F1 scores for the generative RNNG for different proposal models.

	Our samples	Dyer et al. [2016]
Our RNNG	$108.76 \pm 1.52 \quad (107.43)$	$107.80 \pm 1.59 \quad (106.45)$
Dyer et al. [2016]	–	– (105.2)
Kuncoro et al. [2017]	–	– (100.9)

Table 3.5.: Perplexity of the generative RNNG for different proposal models.

samples. The reranking by the generative models still finds better trees than the greedy decoding of the discriminative model, as evidenced by the higher F1 score, and when we evaluate with the proposal samples used in the original work we find that the F1 agrees perfectly with that of Dyer et al. [2016] and is on par with the results reported in Kuncoro et al. [2017].[10]. The difference in perplexity performance is less pronounced, and with our own samples we manage to get close to the perplexity published in Dyer et al. [2016]. The perplexity does lag behind the results reported in Kuncoro et al. [2017], which are the results obtained with the attention-based composition function.

3.5.3. Analysis

We now take a closer look at the approximate inference of the RNNG. The temperature α affects the proposal model and this, together with the number of samples, affects the approximate inference of the generative model. The temperature is used to flatten the proposal distribution to more evenly cover the support of the joint model, with the rationale of thus being closer to the true posterior;[11] the number of samples is used to increase the accuracy of our unbiased MC estimate. The number of samples should particularly garner our attention: the approximate marginalization is compuationally expensive, requiring at least as many forward computations as there are unique samples.[12] The less samples we can get away with the better. To see how these values affect the generative RNNG in practice we perform two experiments: first we evaluate the RNNG with increasing number of samples, obtained with and without annealing; then we use these same samples to estimate the conditional entropy of the proposal models.

For the first experiment we evaluate the generative RNNG as in the above setting, while varying the number of samples, and the temperature α: we let the number of

[10] The proposal samples used by Dyer et al. [2016] are availlable at `https://github.com/clab/rnng`.

[11] After all, the proposal model was trained on a maximum likelihook objective, and we can thus expect it to be too peaked.

[12] For efficiency we already partition the samples into groups of identical samples, where for each group we need to compute just ones and scale by the size of the group.

samples be in $\{10, 25, 50, 75, 100\}$ and let α be in $\{0.8, 1\}$.[13]. To quantify the variability of the proposal model in this we repeat the experiment 10 times. Figure 4.3 shows the results of this experiment for the parsing F1, and 4.4 shows the results for the perplexity. The F1 of the generative model increases with the number of samples, but clearly converges at around 100 samples; the difference between 10 and 100 samples is about 1 F1 on average. This appears to tell us that the generative model finds the better trees at the tails of the proposal distribution—the trees that are only observed, on expectation, given enough samples. There is little difference between the different temperature values when using many samples, but when few samples are used the proposal model that is left unchanged performs best. In the case of perplexity we can see a consistent difference between the values of α, always favoring the proposal model where $\alpha = 0.8$. A striking feature that can be observed in the plot is that the perplexity estimates lie in small regions that decrease with increasing number of samples. The estimate is in fact an *upper bound* on the perplexity, a result of our estimator, the definition of perplexity, and Jensen's inequality.[14] Each of the independent estimates vary litle between each other, but the gap between 10 and 100 samples is a significant 10 nats in perplexity, on avergare. It moreover appears from the trend of the curve that 100 samples is not yet enough to close the bound, and that more samples would still further lower the perplexity—an unfortunate finding given our concerns about computational efficiency.

We move to the second experiment. To quantify the uncertainty in the discriminative proposal model we compute the conditional entropy. The conditional entropy of Y given X is defined as

$$H(Y \mid X) = \sum_{x \in \mathcal{X}} p(x)\, H(Y \mid X = x),$$

where

$$H(Y \mid X = x) = -\sum_{y \in \mathcal{Y}} p(y \mid x) \log p(y \mid x).$$

In our case, the distribution p is given by the discriminative RNNG. Because for this model the sum over $\mathcal{Y}$ cannot be computed efficiently we estimate it with a Monte Carlo estimate as

$$\begin{aligned} H(Y \mid X = x) &= -\,\mathbb{E}[\log p(Y \mid X = x)] \\ &\approx -\frac{1}{K} \sum_{k=1}^{K} \log p(y^{(k)} \mid x), \end{aligned}$$

for which we can use the same samples $y^{(k)}$ drawn from $p(Y \mid X = x)$ as in the above experiments. To estimate the sum over $\mathcal{X}$ we can use the test set $\mathcal{D}$ that, which we

[13]When $\alpha = 1$, the distribution is left unchanched.

[14]We compute $p(x)$ as an expectation $\mathbb{E}[p(x, y)]$, and because $\log \mathbb{E}[p(x, y)] \leq \mathbb{E}[\log p(x, y)]$ by Jensen's inequality we find for our perplexity estimate that $\exp\{-\frac{1}{m} \log \mathbb{E}[p(x, y)]\} \geq \exp\{-\frac{1}{m} \mathbb{E}[\log p(x, y)]\}$.

consider a large sample from the data distribution $p(x)$:

$$H(Y \mid X) \approx \frac{1}{|\mathcal{D}|} \sum_{(x,y)\in\mathcal{D}} H(Y \mid X = x).$$

What this means effectively is that we compute the average negative log-likelihood over all samples, for all sentences in the test set. We repeat this for each of the 10 sets of samples, for the same increasing number of samples. The results are show in figure 4.5. The results are as expected: the estimate converges as the number of samples increases, and the entropy of the proposal model flattened with $\alpha = 0.8$ is much higher than that of the original.

3.6. Related work

Most directly related to the RNNG is the work on generative dependency parsing [Titov and Henderson, 2007, Buys and Blunsom, 2015a,b, 2018] and the work on language modelling via top-down generative parsing models in Roark [2001].

It has been investigated what RNNGs learn about syntax beyond their supervision. Kuncoro et al. [2017] show that the attention mechanism in the composition function learns a type of 'soft' head-rules[15], in which most of the attention weight is put on the item that linguist consider the synactic head of such constituents. Another finding shows that when trained on unlabeledd trees, the RNNG learns representations for constituents that cluster according to their withheld gold label. Additionally, it has been found that RNNGs are much better at a long-distance subject-verb agreement task than LSTMS [Linzen et al., 2016, Kuncoro et al., 2018], which advantage they owe to the composition function that by repeatedly compressing intervening structure decreases the distance between the two words at stake.

It appears also that RNNGs can tell us something about our brains: psycholinguistic research has shown that top-down parsing is a cognitively plausible parsing strategy [Brennan et al., 2016], and recently, RNNGs have been shown to be particularly good statistical predictors for human sentence comprehension [Hale et al., 2018]. In this experiment, the sequential word-probabilities derived from a generative RNNG[16] provide a per-word complexity metric that predict human reading difficulty well. Much better at least than predictions from the word probabilities obtained from a purely sequential RNN language model.

[15] A head is the lexical item in a phrase that determines the syntactic category of that phrase, *cf.*the background.

[16] Obtained by using 'word-synchronous beam-search' [Stern et al., 2017b].

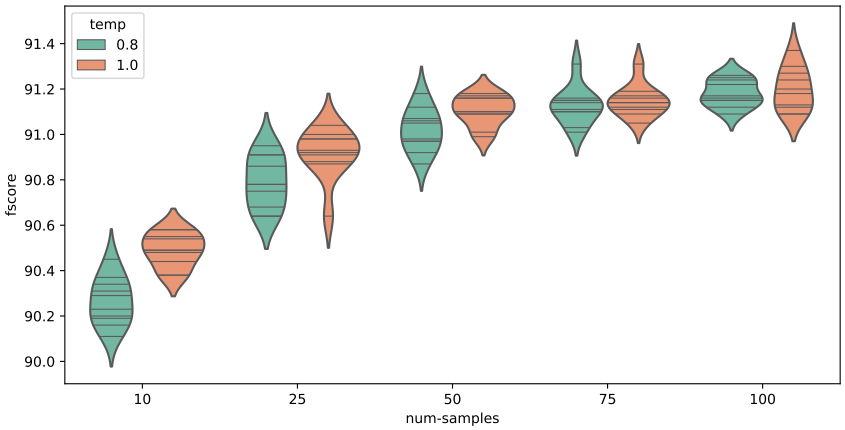


Figure 3.1.: F1 estimated with increasing number of samples, with and without annealed distributions, for 10 independent repetitions. Horizontal lines show values of individual outcomes.

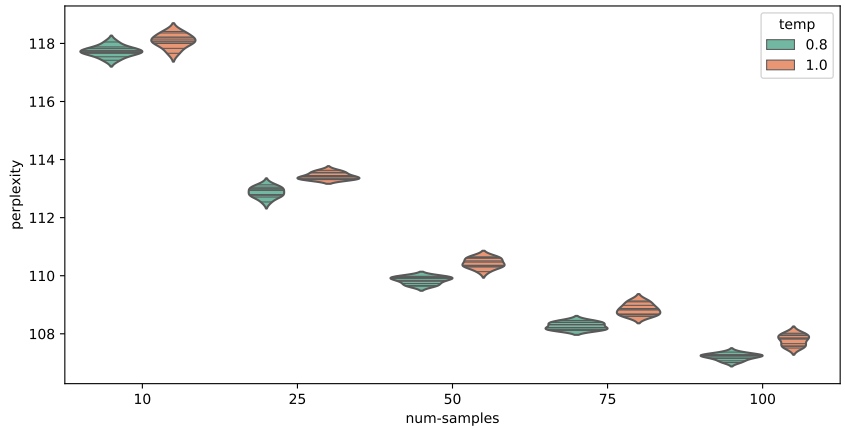


Figure 3.2.: Perplexity estimated with increasing number of samples, with and without annealed distributions, for 10 independent repetitions. Horizontal lines show values of individual outcomes.

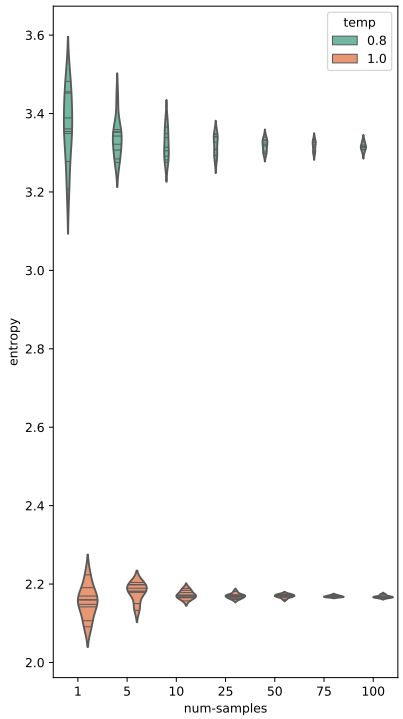


Figure 3.3.: Conditional entropy estimated with increasing number of samples, with and without annealed distributions, for 10 independent repetitions. Horizontal lines show values of individual outcomes.

4. Conditional Random Field parser

In the previous chapter we described how inference with the RNNG can be performed using a discriminative proposal model. In this chapter we introduce an alternative proposal model. We introduce a novel neural Conditional Random Field (CRF) parser. The model is a CRF factored over labeled spans, and is an adaptation of the chart-based parser introduced in Stern et al. [2017a] from margin-based training to global CRF training. The model is defined over normal form trees, but can deal with n-ary trees by binarization using a dummy label, and with unary chains by collapsing them to an atomic label.[1]. This solution is adequate for the supervised training of the CRF, but does pose a challenge for the use of this model as proposal for the RNNG: the implicit binarization causes derivational ambiguity and in turn a subtle mismatch between the set of trees modelled by the CRF and the set of trees modelled by the RNNG. We discuss the effects of the derivational ambiguity and propose solutions at the end of the chapter.

In this chapter:

- We present the model and describe how it is an adaptation into a CRF of the max-margin trained model of Stern et al. [2017a].
- We show how several inference problems of interest—parsing, sampling, and computing the entropy—can be solved exactly and efficiently by deriving specific instances of the inside and outside algorithms.
- We train the model in and show its effectiveness as discriminative parser: with the same number of parameters as the discriminative RNNG it achieves 90.04 F1, surpassing RNNG by more than 1 point.
- We use the trained model as a proposal distribution for the generative RNNG, and show that this does not affect the parsing accuracy, but does affect the perplexity, albeit negatively.
- We desribe the drawbacks of this model as a proposal model for the RNNG, and discuss solutions.

4.1. Model

The model is a span-factored CRF that predicts scores for labeled spans over the sentence using neural networks, which then interact in a tree-structured dynamic program,

[1] *Cf.* figure 2.2 for this normalization process.

giving a compact description of the probability distribution over all parses. This approach combines the efficient exact inference of chart-based parsing, the rich nonlinear features of neural networks, and the global training of a CRF. The factorization enables efficient exact inference alowing for exact decoding and global sampling while the neural features can be complex and can condition on the entire sentence.

Let x be a sentence, and y a tree from $\mathcal{Y}(x)$. We define a function Ψ that assings nonnegative scores $\Psi(x, y)$ and let the probability of a tree be its globally normalized score

$$p(y \mid x) = \frac{\Psi(x, y)}{Z(x)}, \tag{4.1}$$

where

$$Z(x) = \sum_{y \in \mathcal{Y}(x)} \Psi(x, y)$$

is the normalizer, or partition function, that sums over the exponential number of trees availlable for x.

To allow efficient computation of the normalizer, we let the scoring function Ψ factorize over the parts of y. We consider a tree as a set of labeled spans $y_a = (A, i, j)$, indicating that a label A spans the words $\langle x_{i+1}, \ldots, x_j \rangle$, and thus write $y = \{y_a\}_{a=1}^{A}$. The value $\Psi(x, y)$ is then defined as the product

$$\Psi(x, y) = \prod_{a=1}^{A} \psi(x, y_a), \tag{4.2}$$

where nonnegative potentials $\psi(x, y_a)$ score the parts. The function ψ scores each labeled span y_a seperately, but conditional on the entire sentence.

The above model is your typical constituency parsing CRF [Finkel et al., 2008, Durrett and Klein, 2015]. But where those models factorize trees over *anchored rules*, our model the factorizes over *labeled spans*,[2] an approach first taken in Stern et al. [2017a]. This factorization imposes an even stronger independence assumption than that imposed by factorizatin over anchored rules. By factorizing over labeled spans, the potential function ψ has no access to information about the direct substructure under the node, such as the child nodes and their split point, and the function can thus rely less on the (local) tree structure and must thus rely more on the surface features of the input sentence. It will, however, greatly reduce the size of the state-space of the dynamic programs, speeding up training and inference, and the burden of the scoring function will be carried by a rich neural network parametrization. Together this will make the parser fast yet effective, which we will detail in section 4.3 on inference.

[2]Compare table 2.1 for the different representations of a tree.

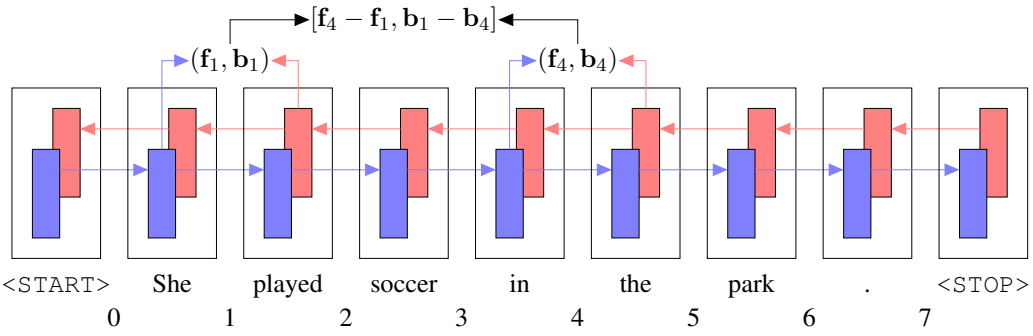


Figure 4.1.: Representation for the span $(1, 4)$ computed from RNN encodings. Figure taken from Gaddy et al. [2018].

4.2. Parametrization

The scoring function ψ is implemented with neural networks following Stern et al. [2017a]. Again, let y_a denote a labeled span (A, i, j) in a tree y, and let $\psi(x, y_a) \geq 0$ be the score of that labeled span for sentence x. These local potentials can only make minimal use of structural information but they can depend on the entire sentence. This suggest the use of bidirectional RNN encodings. Let $\mathbf{f}_i$ and $\mathbf{b}_i$ respectively be the vectors computed by a forward and backward RNN for the word in position i. The representation of the span (i, j) is the concatenation of the difference between the vectors on the endpoints of the span:

$$\mathbf{s}_{ij} = [\mathbf{f}_j - \mathbf{f}_i; \mathbf{b}_i - \mathbf{b}_j]. \tag{4.3}$$

The vector $\mathbf{s}_{ij}$ represents the words x_i^j, and equation 4.3 is illustrated in figure 4.1. The scores for each label in that position are computed from this vector using a feedforward network with output dimension $\mathbb{R}^{|\Lambda|}$, and the score of label A is given by the index corresponding to it:

$$\log \psi(x, y_a) = [\text{FFN}(\mathbf{s}_{ij})]_A, \tag{4.4}$$

where we pretend that A doubles as an index. This architecture is rightly called minimal, but it works surprisingly well: Stern et al. [2017a] also experiment with more elaborate functions based on concatenation of vectors (a strict superset of the minus approach) and biaffine scoring (inspired by Dozat and Manning [2016]), but these functions improve marginally, if they do at all.

4.3. Inference

Because our model is span-factored it allows efficient inference. In this section we describe efficient solutions to four related problems:

- Compute the normalizer $Z(x) = \sum_{y \in \mathcal{Y}(x)} \prod_{a=1}^{A} \psi(x, y_a)$.
- Find the best parse $\hat{y} = \arg\max_y p(y \mid x)$
- Sample a tree $Y \sim P(Y \mid X = x)$.
- Compute the entropy $H(Y \mid X = x)$ over parses for x.

These problems can be solved by instances of the *inside algorithm* and *outside algorithm* [Baker, 1979] with differentent semirings, an insight we take from semiring parsing [Goodman, 1999]. In the following derivations we will make use of the notion of a *weighted hypergraph* as a compact representation of all parses and their scores [Gallo et al., 1993, Klein and Manning, 2004], and use some of the ideas and notation of *semiring parsing* [Goodman, 1999, Li and Eisner, 2009]. First we describe the structure of the parse forest specified by our CRF parser, and then derrive the particular form of the inside and outside recursions for this hypergraph from the general formulations. We refer the reader to appendix B for background on these ideas, and the introduction of the notation.

4.3.1. Weighted parse forest

The the CRF parser defines a hypergraph $G = (\mathcal{V}, \mathcal{E})$ with the following structure.

The set $\mathcal{V}$ is defined relative to the sentence x, and contains the invidual words of a sentence x, together with all possible labeled spans over that sentence:

$$\mathcal{V} = \Big\{ x_i \;\Big|\; 1 \leq i \leq n \Big\} \cup \Big\{ (A, i, j) \;\Big|\; A \in \Lambda, 0 \leq i < j \leq n \Big\} \cup \Big\{ (S^\dagger, 0, n) \Big\},$$

where $(S^\dagger, 0, n)$ is a designated root node. The dependence on x can be made explicit by writing $\mathcal{V}(x)$.

The set of hyperedges $\mathcal{E} \subseteq 2^{\mathcal{V}} \times \mathcal{V}$ specifies all the ways that adjacent constituents can be combined to form a larger constituent, under a particular grammar. Because we (implicitly) assume a normal form grammar that contains *all* possible productions, the set of hyperedges is particularly regular: the set $\mathcal{E}$ contains all edges that connect nodes (B, i, k) and (C, k, j) at the tail with (A, i, j) at the head for all $0 \leq i < k < j \leq n$, all edges that connect x_i to $(A, i, i+1)$, and all labels can appear in the top node, with the exception of the dummy label $\varnothing$:

$$\begin{aligned}
\mathcal{E} = &\left\{ \left\langle \Big\{ (B, i, k), (C, k, j) \Big\}, (A, i, j) \right\rangle \;\middle|\; A, B, C \in \Lambda,\ 0 \leq i < k < j \leq n \right\} \\
&\cup \left\{ \left\langle \{x_i\}, (A, i-1, i) \right\rangle \;\middle|\; A \in \Lambda,\ 1 \leq i \leq n \right\} \\
&\cup \left\{ \left\langle \{(A, 0, n)\}, (S^\dagger, 0, n) \right\rangle \;\middle|\; A \in \Lambda \setminus \{\varnothing\} \right\}
\end{aligned}$$

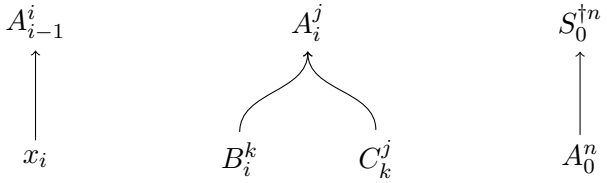


Figure 4.2.: The edge-types making up the hypergraph. The edges are instatiated for all $A, B, C \in \Lambda$ and all $0 \leq i < j \leq n$. Only for the rightmost edge holds the restriction $A \neq \varnothing$, to ensure that the dummy label cannot be the top node.

The three kind of edges that make up $\mathcal{E}$ are illustrated in figure 4.2. A tree is a set of nodes $y \subseteq \mathcal{V}(x)$, and the parse forest a set of trees $\mathcal{Y}(x) \subseteq 2^{\mathcal{V}(x)}$.

We connect a semiring $\mathcal{K}$ to the hypergraph by defining the weight function as $\omega : \mathcal{E} \to \mathbb{K}$, and by accumulating the weights with its binary operations. The function ω that assigns weights to the edges is given by either the function ψ or $(\log \circ \psi)$, depending on the semiring used. Because of this association, the function ω has a very particular property: the function effectively depends only on the *head* of the edge. Given edges $e = \langle \{u, w\}, v \rangle$ and $e' = \langle \{u', w'\}, v \rangle$ for $u \neq u'$ and $w \neq w'$, their weights are equal: $\omega(e) = \omega(e')$. For this reason we write $\omega(v)$ instead.[3] This fact will allow us to greatly simplify the recursions that follow. Additionally, this means that instead of computing independent scores for each of the $O(n^3|\Lambda|^3)$ edges, we only need to compute scores for the $O(n^2|\Lambda|)$ vertices. For a scoring function like a neural network, for which computation can be relatively expensive, this will make a significant difference.

With this structure in place, we are ready to derive the form of the inference algorithm particular to this structure.

4.3.2. Inside recursion

The inside recursion computes quantities $\alpha(A, i, j)$ for all labels $A \in \Lambda$ and all spans $0 \leq i < j \leq n$. The quantity computed depends on the semiring used. In this section we derive the inside recursion specific to our hypergraph from the general result given.

Let $\mathcal{K}$ be some semiring with binary operations $\oplus$ and $\otimes$ and identity elements $\bar{0}$ and $\bar{1}$. The inside recursion is given by the formula [Goodman, 1999]

$$\alpha(v) = \begin{cases} \bar{1} & \text{if } I(v) = \varnothing, \\ \bigoplus_{e \in I(v)} \omega(e) \otimes \bigotimes_{u \in T(e)} \alpha(u) & \text{otherwise.} \end{cases}$$

Here $I(v) \subseteq E$ is the set of edges incoming at node v, and $T(e) \subseteq V$ is the set of nodes in the tail of edge e.

[3] Thus implicitly defining $\omega : \mathcal{V} \to \mathbb{K}$.

At a node $v = (A, i, i+1)$ that spans one word x_i, the inside value is just the weight of the single edge incoming from that word:

$$\alpha(A, i, i+1) = \omega(A, i, i+1) \otimes \alpha(x_i) = \omega(A, i, i+1), \tag{4.5}$$

for $A \in \Lambda$, for all $0 \leq i < n$. We used the fact that $\alpha(x_i) = \bar{1}$, which follows from the fact that there are no arrows incoming at x_i.

For a general node $\alpha(A, i, j)$, $j > i+1$, we observe that all the incoming edges have at the tail the nodes (B, i, k) and (C, k, j), for all $B, C \in \Lambda$ and $i < k < j$. The sum over edges thus reduces to independent sums over B, C, and k, and the product over the inside values at the tail reduces to the product of values $\alpha(B, i, k)$ and $\alpha(C, k, j)$. The form of ω allows us to rewrite this greatly as

$$\begin{aligned}
\alpha(A, i, j) &= \bigoplus_{B \in \Lambda} \bigoplus_{C \in \Lambda} \bigoplus_{k=i+1}^{j-1} \omega(A, i, j) \otimes \alpha(B, i, k) \otimes \alpha(C, k, j) \\
&= \omega(A, i, j) \otimes \bigoplus_{k=i+1}^{j-1} \bigoplus_{B \in \Lambda} \alpha(B, i, k) \otimes \bigoplus_{C \in \Lambda} \alpha(C, k, j) \\
&= \omega(A, i, j) \otimes \bigoplus_{k=i+1}^{j-1} \sigma(i, k) \otimes \sigma(k, j),
\end{aligned} \tag{4.6}$$

where we've introduced the notational abbreviation

$$\sigma(i, j) = \bigoplus_{A \in \Lambda} \alpha(A, i, j).$$

Looking at 4.6 we can see the marginalized values $\sigma(i, j)$ are all that are needed for the recursion. This suggest simplifying the recursion even further as

$$\begin{aligned}
\sigma(i, j) &= \bigoplus_{A \in \Lambda} \alpha(A, i, j) \\
&= \left[\bigoplus_{A \in \Lambda} \omega(A, i, j) \right] \otimes \left[\bigoplus_{k=i+1}^{j-1} \sigma(i, k) \otimes \sigma(k, j) \right],
\end{aligned} \tag{4.7}$$

where we put explicit brackets to emphasize that independence of the subproblems of labeling and splitting.

These values can be computed by visiting the nodes of the parse forest in topological order. This order ensures that the quantities in the tail of an edge will have been computed when the value at the node at the head is at turn. Given the highly regular form of the parse forest, this order boils down to something quite simple: the nodes are visited by increasing width of the span; the order in which the starting-points are visited does not matter because these nodes are not connected among each other.[4]. We choose to visit the starting points from left to right, that is, from 0 to n.

[4] *Cf.* figure B.1: the width of the span determines the vertical level of the node in the parse forest; for a fixed width, the label and startpoint are all unconnected along this vertical level, and thus independent.

4.3.3. Outside recursion

The outside algorithm computes the quantities $\beta(A, i, j)$ for all labels $A \in \Lambda$ and all spans $0 \leq i < j \leq n$. The general recursion is given by:

$$\beta(v) = \begin{cases} \bar{1} & \text{if } O(v) = \varnothing, \\ \bigoplus_{e \in O(v)} \omega(e) \otimes \beta(H(e)) \otimes \bigotimes_{\substack{w \in T(e) \\ w \neq u}} \alpha(w) & \text{otherwise.} \end{cases}$$

Here, $O(v) \subseteq E$ is the set of edges outgoing from v, *i.e.* the edges for which v is in the tail, and define $H(e) \in V$ as the node at the head of edge e.

The only node without outgoing edges is the root node, and thus

$$\beta(S^\dagger, 0, n) = \bar{1}.$$

To compute $\beta(A, i, j)$ in the general case we need to sum over all outgoing edges. These come in two kinds: either (A, i, j) combines with (C, j, k) to form constituent (B, i, k); or (A, i, j) combines with (C, k, i) to form constituent (B, k, j). This corresponds to the following expression, that we can simplify by making use of the properties of ω:

$$\begin{aligned} \beta(A, i, j) &= \bigoplus_{B \in \Lambda} \bigoplus_{C \in \Lambda} \bigoplus_{k=1}^{i-1} \omega(B, k, j) \otimes \alpha(C, k, i) \otimes \beta(B, k, j) \\ &\quad \oplus \bigoplus_{B \in \Lambda} \bigoplus_{C \in \Lambda} \bigoplus_{k=j+1}^{n} \omega(B, i, k) \otimes \beta(B, i, k) \otimes \alpha(C, j, k) \\ &= \bigoplus_{k=1}^{i-1} \left[\bigoplus_{B \in \Lambda} \omega(B, k, j) \otimes \beta(B, k, j) \right] \otimes \left[\bigoplus_{C \in \Lambda} \alpha(C, k, i) \right] \\ &\quad \oplus \bigoplus_{k=j+1}^{n} \left[\bigoplus_{B \in \Lambda} \omega(B, i, k) \otimes \beta(B, i, k) \right] \otimes \left[\bigoplus_{C \in \Lambda} \alpha(C, j, k) \right] \\ &= \bigoplus_{k=1}^{i-1} \sigma'(k, j) \otimes \sigma(k, i) \oplus \bigoplus_{k=j+1}^{n} \sigma'(i, k) \otimes \sigma(j, k) \end{aligned}$$

where

$$\begin{aligned} \sigma(i, j) &= \bigoplus_{A \in \Lambda} \alpha(A, i, j), \\ \sigma'(i, j) &= \bigoplus_{A \in \Lambda} \omega(A, i, j) \beta(A, i, j). \end{aligned}$$

These values can be computed by visiting the nodes of the parse forest in *reverse* topological order, exactly the opposite of the inside recursion. This order ensures that

the quantities in the head of an edge will have been computed when the values in the tail are at turn. Again, this this order boils down to something quite simple: the nodes are visited based by decreasing width of their span, and again the order of the start-points are free.

4.3.4. Solutions

Equiped with the two recursions and a handful of semirings we can provide the solutions promised at the outset of this section.

Normalizer When we intantiate the inside recursion with the real semiring, the value of α at the root is the normalizer:

$$\alpha(\mathsf{S}^\dagger, 0, n) = Z(x),$$

and when we instantiate the inside recursion with the log-real semiring we obtain the log-normalizer

$$\alpha(\mathsf{S}^\dagger, 0, n) = \log Z(x).$$

Parse To find the viterbi tree $\hat{y} = \arg\max_y p(y \mid x)$ and its probability $p(\hat{y} \mid x)$ we use the Viterbi semirings (*cf.* examples B.3.7 and B.3.8 in appendix B). We take equation 4.7 and use the Viterbi semiring operations to derive that the value of the best subtree spanning words i to j is given by

$$\sigma(i, j) = \max_A[\log \psi(A, i, j)] + \max_k[\sigma(i, k) + \sigma(k, j)]. \tag{4.8}$$

The value $\log \Psi(x, \hat{y})$ is then given by $\sigma(0, n)$, and can be normalized with to give the probability

$$\log p(\hat{y} \mid x) = \sigma(0, n) - \log Z(x). \tag{4.9}$$

The best label and splitpoint $\hat{A}$ and $\hat{k}$ for the span (i, j) are obtained by using the argmax:

$$\hat{A} = \arg\max_A \log \psi(A, i, j) \tag{4.10}$$

$$\hat{k} = \arg\max_k \sigma(i, k) + \sigma(k, j), \tag{4.11}$$

and the best tree $\hat{y}$ is found by following back from the root down to the leaves the best splits and labels.

Sample Unbiased samples from the model can be obtained by recursively sampling incoming edges, starting at the root node $(S^\dagger, 0, n)$, ending at the word nodes. The probability of an edges is proportional to the weight of all the trees under that edge. This is precicely what is represented by the inside value α computed in the real-semiring. At node $v = (A, i, j)$ the probability of edge $e = \langle \{u, w\}, v \rangle$, with $u = (B, i, k)$ and $w = (C, k, j)$, is thus

$$\begin{aligned} P(E = e \mid V = v) &= \frac{\omega(e) \otimes \bigotimes_{u \in T(e)} \alpha(u)}{\alpha(v)} \\ &= \frac{\psi(A, i, j)\, \alpha(B, i, k)\, \alpha(C, k, j)}{\alpha(A, i, j)}. \end{aligned} \tag{4.12}$$

Once an edge is sampled, the process is repeated at the nodes in the tail of that edge. We stop when we reach the nodes that span individual words. The sampled edges together then are guaranteed to form a tree. From a compuational point of view, this sampling is very efficient: the scores for the labeled spans need to be computed only once, followed by a single run of the inside and outside algorithm. Beyond this point, no costly computation is required. In our case, where the scores are predicted by a neural network, this means that we need to perfom just a single forward pass. Compare this with a sequential model such as the discriminative RNNG, where each sample requires an separate forward pass.

Entropy To compute the entropy $H(Y \mid X = x)$ we need to first introduce the notion of a *node maginal*. The marginal of a node $v = (A, i, j)$ in a hypergraph is the probability that it occurs in a tree y for the sentence x, according to the probability distribution p over trees. The node marginal $\mu(v)$ is defined as the expectation

$$\begin{aligned} \mu(v) &\triangleq \mathbb{E}[\mathbf{1}_{\{v \in Y\}}] \\ &= \sum_{y \in \mathcal{Y}(x)} p(y \mid x)\, \mathbf{1}_{\{v \in y\}}\,. \end{aligned} \tag{4.13}$$

This can be computed from the inside and outside values computed in the real semiring as

$$\mu(v) = \frac{\alpha(A, i, j)\, \beta(A, i, j)}{Z(x)}, \tag{4.14}$$

a result from [Goodman, 1999]. This can also be seen by noting that the product of $\alpha(A, i, j)$ and $\beta(A, i, j)$ is the sum over all trees that contain the node v, and $Z(x)$ the sum over all trees in general.

The entropy, then can then be written in terms of these marginals:

$$\begin{aligned} H(Y \mid X = x) &= -\sum_{y \in \mathcal{Y}(x)} p(y \mid x) \log p(y \mid x) \\ &= \log Z(x) - \sum_{y \in \mathcal{Y}(x)} p(y \mid x) \sum_{v \in y} \log \psi(x, v) \\ &= \log Z(x) - \sum_{y \in \mathcal{Y}(x)} p(y \mid x) \sum_{v \in \mathcal{V}(x)} \mathbf{1}_{\{v \in y\}} \log \psi(x, v) \\ &= \log Z(x) - \sum_{v \in \mathcal{V}(x)} \log \psi(x, v) \sum_{y \in \mathcal{Y}(x)} \mathbf{1}_{\{v \in y\}} \, p(y \mid x) \\ &= \log Z(x) - \sum_{v \in \mathcal{V}(x)} \log \psi(x, v) \mu(v) \end{aligned} \tag{4.15}$$

4.4. Traning

The training objective is to maximize the likelihood of the training data

$$\mathcal{L}(\theta) = \sum_{(x,y) \in \mathcal{D}} \log p_\theta(y \mid x)$$

with respect to the model parameters θ.

4.4.1. Objective

Writing out the objective for a single example as

$$\log p_\theta(y \mid x) = \log \Psi(x, y) - \log Z(x)$$

reveals that the maximization of this value decomposes as two separate optimization-problems: to maximize the log-score of the example tree $\log \Psi(x, y)$, whilst minimizing the log of the total weight of the parse forest $\log Z(x)$. The solution is thus to move probability mass onto the gold tree y, and away from *all other trees*. Because the log-score of the tree decomposes as a sum of log-scores of the nodes that it comprises, the objective is effectively to *increase* the scores of the labeled spans that make up the gold tree, and *decrease* the scores of all other labeled spans not observed. Another effect of this factorization into spans is that such an update increases not only the probability of the gold tree, but the probability of all trees that share substantial substructure. It is in this precise sense that we mean that the model learns about all substructures.

We rely on automatic differentiation to compute the derivatives, but note that in principle it is possible to efficiently combine the computation of the inside and outside values with their derivatives, a fact demonstrated in [Li and Eisner, 2009] and [Eisner, 2016], and implemented in [Kim et al., 2017].[5]

[5] Amongts others, we do not take because it would require us to implement custom gradient computations in our toolkit of choice Dynet Neubig et al. [2017a], which is far from trivial.

It is fruitful to see how our global objective differs from the margin-based objective in [Stern et al., 2017a]. The margin-based objective is to maximize the difference, or 'margin', between the score of the gold tree and the highest scoring, incorrect tree. Given a sentence x the model computes the predicted tree $\hat{y}$. If the predicted tree equals the gold tree, $\hat{y} = y$, then no changes to the model parameters are made. Otherwise, the model parameters are updated to maximize the difference between their scores

$$\log \Psi(x, y) - \log \Psi(x, \hat{y}),$$

maximizing the score of the correct tree, and minimizing the score of the predicted tree.[6] This objective shows a striking similarity with our CRF objective, with one very particular difference. The goal is still to maximize the score of the gold tree. But the minimization that in the CRF objective concerns all possible trees through $Z(x)$, in this objective concerns just the single tree $\hat{y}$ that was incorrectly predicted. The scores of all other nodes not observed in either tree are not affected directly.

4.4.2. Speed and complexity

The model is slow. Here we will describe how slow it is, and what can be done about it.

- During training the computation time is dominated by two computations: the forward pass with the neural networks that obtains the node scores, and the time complexity of the inside algorithm. The complexity of the and how it depends on both label size and sentence length
- Describe how slow the model is to train of CPU (use Lisa times): around 7 hours per epoch, so around 15 days for convergence (around 50 epochs).
- Describe how we can speed up linearly by pruning the labelset: we can remove 70% of labels while keeping 98% of the training sentences. These labels are mostly very rare unary chains. The speedup we get from this is linear!
- Our model is X times slower than the model of Stern et al. [2017a]. This difference bust be in the inside algorithm. The inside algorithm used by us and the viterbi algorithm used by Stern et al. [2017a] have the same structure and time complexity. The difference must be in the computation graph built by either computation. The viterbi algorithm builds a sparse computation graph, connecting only nodes that are in a single tree, whereas the computation graph built by the inside algorithm is dense with all trees.

[6] The objective reported in the paper is minimizing the hinge loss $\max\Big(0, 1 - \log \Psi(x, y) + \log \Psi(x, \hat{y})\Big)$, but the above is what the actual implementation comes down to.

4.5. Experiments

This section reports the results of the experiments performed with the CRF parser. We first train the model on the Penn Treebank and show that it is a strong parser. The CRF obtains a higher F1 than the discriminative RNNG when trained with the same number of parameters; an even higher F1 is obtained when or models is trained with the same hyperparameters as the model in Stern et al. [2017a]. We then use the CRF as an alternative proposal model for the generative RNNG and perform analyses similar to those in chapter 3. This use of the CRF leads to some theoretical challenges. We note these, but discuss them in detail in the next section.

4.5.1. Setup

We train two versions of the model on the Penn Treebank: a small version and a large version. The smaller model allows us to compare the CRF parser to the discriminative RNNG, while the larger model allows us to compare it to the parser of Stern et al. [2017a]. The smaller model has dimensions to match the number of parameters in the discriminative RNNG from chapter 3 and is trained in the same way. For the larger model we follow the dimensions and optimization details of Stern et al. [2017a].

For both models the embeddings are of dimension 100, and the LSTMs have 2 layers. For the smaller model the dimension of forward and backward the LSTM is 128, and the feedforward network has one hidden layer with dimension 256. For the larger model we follow the hyperparameters of Stern et al. [2017a]: the LSTMs have dimesion 250, just as the feedforward network. Given these dimensions, the total number of parameters in the small model is around 0.75M and around 2.5M for the large model (the exact numbers are in table A.1). The small CRF is comparable in size with the discriminative RNNG (around 0.8M parameters) and less than a third in size of the larger model. The smaller model is optimized exactly as the discriminative RNNG: SGD with learning rate 0.1, and dropout of 0.2. For the larger model we use Adam [Kingma and Ba, 2014] with 0.001 and dropout of 0.4, exactly following Stern et al. [2017a].

4.5.2. Results

The evaluation setup is the same as in the previous chapter. We train 10 models from different random seeds, and we report mean and standard deviations, as well as scores of the best model selected by development F1.

	Small		Large		Stern et al. [2017a]	
F1	89.94 ± 0.12	(90.04)	90.31 ± 0.15	(90.43)	–	(91.79)

Table 4.1.: F1 scores for the CRF.

The results are shown in 4.1. Both models perform well, achieving on average 89.94 F1 for the smaller model and 90.31 F1 for the larger model. The small difference in F1

between the two is surprising given difference in size and optimization, and speaks for the relative robustness of our model with respect to these choices. Our best small CRF (90.04 F1) surpasses the best discriminative RNNG (88.58) by almost 2.5 F1, but our best large CRF (90.43) performs below the model of Stern et al. [2017a] (91.79). Just as for the RNNG, we believe this can be attributed to the absence of tags. However, we can note that—in case this is true—the negative impact is much greater for the RNNG (around 3.1 F1 lower) than in the CRF (around 1.4 F1 lower).

4.5.3. Proposal distribution

We have introduced the CRF parser as an alternative proposal model for the generative RNNG, and we now evaluate the CRF in this role. We repeat the same inference procedure as in chapter 3: we sample 100 trees from the small CRF with the highest development F1, and use these to evaluate the same generative RNNG as before. The results are shown in tables 4.2 (F1) and 5.1 (perplexity), and include the previous results for comparison.

	RNNG	CRF	Dyer et al. [2016]
Our RNNG	91.07 ± 0.1 (91.12)	91.02 ± 0.05 (91.04)	93.32 ± 0.1 (93.32)
Dyer et al. [2016]	–	–	– (93.3)

Table 4.2.: F1 scores for the generative RNNG for different proposal models.

	RNNG	CRF	Dyer et al. [2016]
Our RNNG	108.76 ± 1.52 (107.43)	117.79 ± 2.1 (116.28)	107.80 ± 1.59 (106.45)
Dyer et al. [2016]	–	–	– (105.2)
Kuncoro et al. [2017]	–	–	– (100.9)

Table 4.3.: Perplexity on the PTB of the generative RNNG, for different proposal models.

The results for the F1 are virtually the same. Although the CRF is the better parser by F1, both are equivalent with respect to generative RNNG. On average, however, because the discriminative RNNG proposals performs better on the selected model. A real difference can be observed in the perplexity, with a difference of over 9 nats in favour of the RNNG.

4.5.4. Analysis

We perform the same analyses of the approximate inference as in chapter 3, this time with the CRF as proposal model. The results of the experiment on aproximate inference are shown in figure 4.3 (F1) and 4.4 (perplexity), where they are compared to the results of the annealed RNNG. With respect to parsing F1, the CRF is the better choice when using few samples, but for this difference disappears for larger number of samples.

Figure 4.5 shows the estimation of the conditional entropy, now including the results for the CRF. The estimates of the CRF converge to the true conditional entropy, which can be computed exactly and equals 2.84.[7] Furthermore we can note that the conditional entropy of the CRF is higher than that of the discriminative RNNG when not annealed. This could indicate that the CRF maintains a higher number of alternative parses per sentence on average than the RNNG. Another plausible explanation is that the higher entropy is caused by the derivational ambiguity: the higher entropy reflects that each tree has multiple derivations.

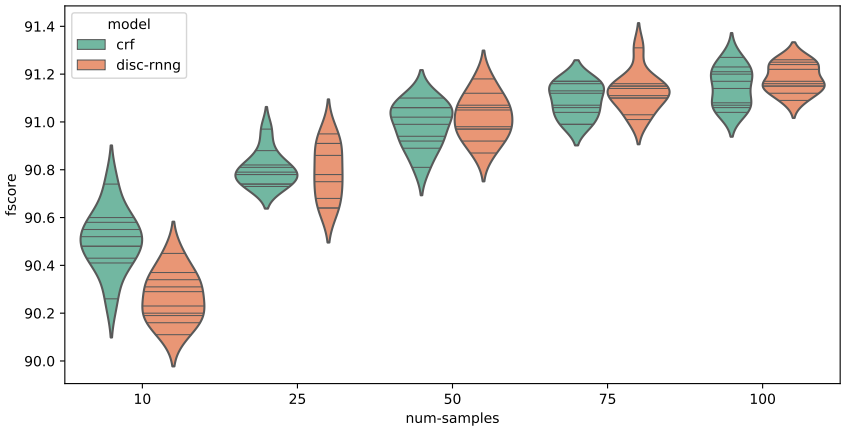


Figure 4.3.: F1 estimated with increasing number of samples, with CRF and discriminative RNNG annealed with $\alpha = 0.8$, for 10 independent repetitions.

[7] To be precise, the values $H(Y \mid X = x)$ can be computed exactly, the value $H(Y \mid X)$ is still estimated by a mean over the test set.

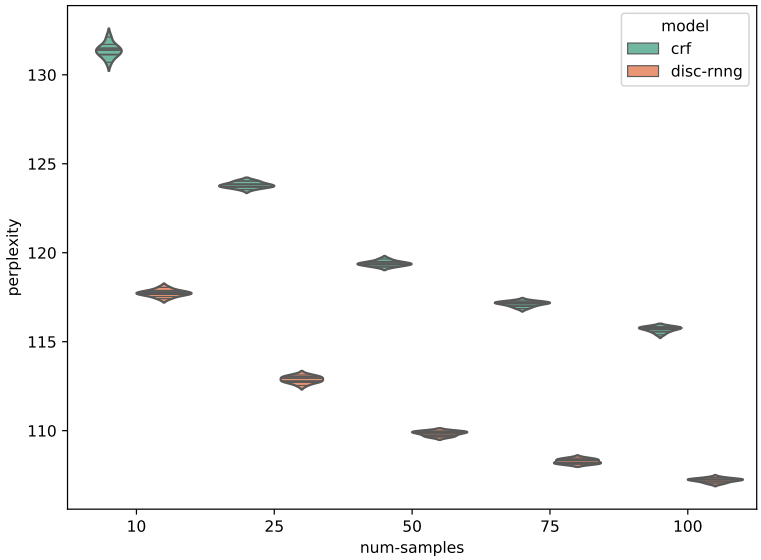


Figure 4.4.: Perplexity estimated with increasing number of samples, with CRF and discriminative RNNG annealed with $\alpha = 0.8$, for 10 independent repetitions.

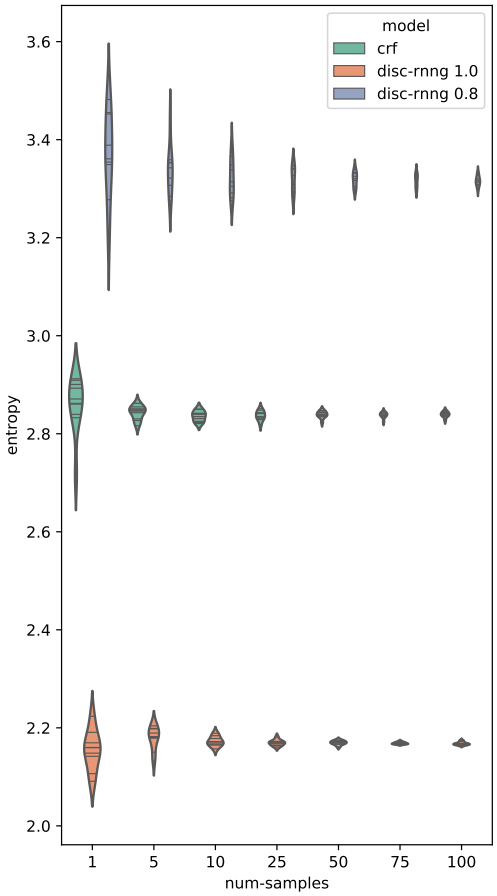


Figure 4.5.: Conditional entropy estimated with increasing number of samples.

4.6. Trees and derivations

The use of the CRF as a proposal distribution brings to light a subtle issue: by our choice of binarization we have introduced derivational ambiguity, and this affects the approximate marginalization. In this section we will describe the issue, which we characterize as the difference between the binary *derivations* and the *trees* they collapse to. We analyze the impact of this on the approximate marginalization, and then propose solutions. We will return to this difference in chapter 6, where the computation of the entropy plays a central role.

4.6.1. Derivational ambiguity

The way our CRF deals with binarization and its inverse introduces the problem derivational ambiguity: many binary derivation d modelled by the CRF as distinct map to the same tree y when the dummy node is collapsed. In other words: we map a treebank tree to a single normal form derivation, but many normal form derivations map to a that treebank tree. For example, the two normal form trees in 4.6 both collapse to the same tree when dummy labels $\varnothing$ are collapsed:

(S (NP The other hungry cat) (VP meows) .).

However, when going the other way—applying the normal form transformation to the above tree—we always obtain the left tree, and never the right tree. Let $f : \mathcal{Y}(x) \to \mathcal{D}(x)$ be the transformation that we defined that produces normal form trees from treebank trees, and let $g : \mathcal{D}(x) \to \mathcal{Y}(x)$ be the transformation that takes a normal form tree and collapses the dummy nodes. Then f is a bijection, but g is not: for any tree y the preimage of g is the set of derivations $g^{-1}(y) = D \subseteq \mathcal{D}(x)$ that collapse to the same tree. And generally, $|D| > 1$. This is the issue at hand: going from derivations to trees causes derivational ambiguity.

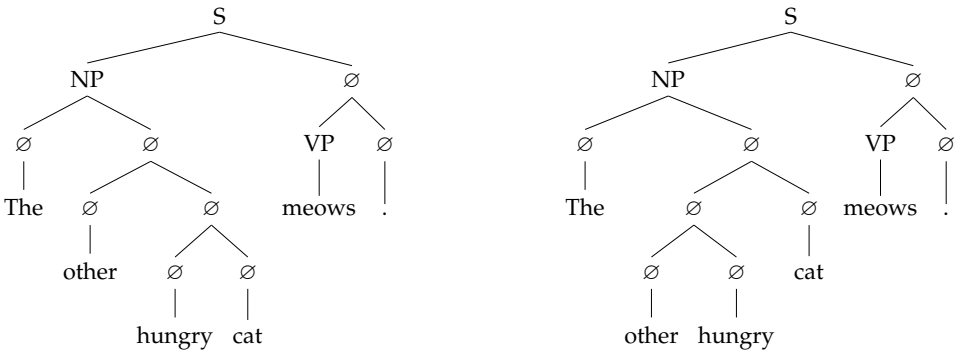


Figure 4.6.: Two normal form derivations that collapse to the same tree.

4.6.2. Consequences

The direct consequence of the derivational ambiguity is that our CRF actually defines a distribution over $\mathcal{D}(x)$, and over $\mathcal{Y}(x)$ only through by summing over the derivations that collapse to it. Let y be a general tree such as is modelled by the RNNG. Our CRF assigns probabilities to all $d \in f^{-1}(y)$ and defines a distribution over $\mathcal{Y}(x)$ through

$$p(y \mid x) = \sum_{d \in f^{-1}(y)} p(d \mid x).$$

As a consequence $p(d \mid x) \leq p(y \mid x)$ for all of the derivations d that collapse to y. This has three direct consequences:

- We overestimate the marginal probability:

$$\begin{aligned}
\frac{1}{K} \sum_{i=1}^{K} \frac{p_\theta(x, y^{(i)})}{q_\lambda(d^{(i)} \mid x)} &\geq \frac{1}{K} \sum_{i=1}^{K} \frac{p_\theta(x, y^{(i)})}{\sum_{d \in f^{-1}(y^{(i)})} q_\lambda(d \mid x)} \\
&= \frac{1}{K} \sum_{i=1}^{K} \frac{p_\theta(x, y^{(i)})}{q_\lambda(y^{(i)} \mid x)} \\
&\approx \mathbb{E}_q \left[\frac{p_\theta(x, y)}{q_\lambda(y \mid x)} \right] = p_\theta(x).
\end{aligned}$$

- Viterbi returns the best derivation, not necessarily the best tree. In other words, if $\hat{d} = \arg\max_d p(d \mid x)$ then $g(\hat{d}) \neq \arg\max_y p(y \mid x)$ in general. This is a situation similar to decoding in latent variable PCFGs [Petrov et al., 2006], which is known to be intractable [Sima'an, 2002].

- The entropy is over derivations instead over trees, and it is not clear how one relates to the other.

The derivational ambiguity does not affect the samples: the probability that we draw y is precisely the probability that we draw any d in $f^{-1}(y)$.

4.6.3. Solutions

We propose two solutions to this problem:

1. We can dispense with the dummy label altogether, and instead alter our CRF so that it can deal directly with m-ary trees, for any order m. This means that we add all hyperedges with more than 2 children. The general formulation of the inside and outside recursions remain unaltered, but the specific form becomes less efficient because we now need to deal with a sum over all possible partitions of the span under a node.

2. We keep the dummy label, but prune the hyperforest to only contain derivations that are reachable by the normal form transformation. This means that we assign probability zero to all trees that do not correspond to the cannonical normal form that we commited to by choosing the transformation. Because this transformation is very regular, we can easily incorporate this pruning in the inside and outside algorithms.

We now describe either solution, starting with the solution of making the CRF deal directly with m-ary trees.

4.6.4. Unrestricted parse forest

One solution, as noted, is to make the parse-forest deal with m-ary trees directly. This requires that for each noterminal node, we add incoming edges that contain more than two children in the tail. This needs to be done for each permissible number of children—determined by the width of the span—and for all possible partitions of that size of the span.

More formally, let (A, i, j) be a node with $i + 1 < j$, so that it can expand to other noterminal nodes. Then let $i = k_0 < k_1 < k_2 < \cdots < k_m = j$ be a partition of the discrete interval $[i, j]$ into m subspans. Since a span must have a width of at least 1, m can be at most $j - i$ Letting $B_1, \ldots, B_m$ be labels for those subspans we can form an edge that connects the set

$$\Big\{(B_1, k_0, k_1), (B_2, k_1, k_2), \ldots, (B_m, k_{n-1}, k_m)\Big\}$$

at the tail to the node (A, i, j) at the head. Adding *all* such nodes to the parse forest for each m will allow us to deal with trees of unrestricted arity directly.

The generality of the semiring formulations of the inside and outside algorithms, allows us to seamlesly apply them to this new parse forest. If we let $v = (A, i, j)$ such that $I(v) \neq \varnothing$ we can derive the inside value $\alpha(v)$ as

$$\begin{aligned}
\alpha(v) &= \bigoplus_{e \in I(v)} \omega(e) \otimes \bigotimes_{u \in T(e)} \alpha(u) \\
&= \bigoplus_{m=2}^{j-i} \bigoplus_{k_0 < k_1 < \cdots < k_m} \bigoplus_{B_1 \in \Lambda} \cdots \bigoplus_{B_m \in \Lambda} \omega(A, i, j) \otimes \bigotimes_{l=1}^{n} \alpha(B_l, k_{l-1}, k_l) \\
&= \omega(A, i, j) \otimes \bigoplus_{m=2}^{j-i} \bigoplus_{k_0 < k_1 < \cdots < k_m} \bigotimes_{l=1}^{n} \bigoplus_{B \in \Lambda} \alpha(B, k_{l-1}, k_l) \\
&= \omega(A, i, j) \otimes \bigoplus_{m=2}^{j-i} \bigoplus_{k_0 < k_1 < \cdots < k_m} \bigotimes_{l=1}^{n} \sigma(k_{l-1}, k_l),
\end{aligned}$$

where logic that allows us write

$$\bigoplus_{B_1 \in \Lambda} \cdots \bigoplus_{B_m \in \Lambda} \bigotimes_{l=1}^{n} \alpha(B_l, k_{l-1}, k_l) = \bigotimes_{l=1}^{n} \bigoplus_{B \in \Lambda} \alpha(B, k_{l-1}, k_l)$$

is the same as in the binary case, but generalized to m terms.

For the outside value $\beta(v)$ we can follow a similar derivation. Let

$$0 \leq k_0 < \cdots < k_a = i < j = k_{a+1} < \cdots < k_m \leq n$$

be a partition of the discrete interval $[k_0, k_m]$ into m pieces, one of which is $[i, j]$, namely the interval $[k_a, k_{a+1}]$. This partition represents one way in wich the interval $[i, j]$ can be completed with $m - 1$ other intervals into an interval $[k_0, k_m]$. With this in place, we can derive the outside recursion as

$$\begin{aligned}
\beta(v) &= \bigoplus_{e \in O(v)} \omega(w) \otimes \beta(H(e)) \otimes \bigotimes_{\substack{w \in T(e) \\ w \neq u}} \alpha(w) \\
&= \bigoplus_{m=2}^{n-j+i} \bigoplus_{k_0=0}^{i-1} \bigoplus_{k_m=j}^{n} \bigoplus_{k_0 < \cdots < k_a < k_{a+1} < \cdots < k_m} \bigoplus_{B \in \Lambda} \\
&\qquad \omega(A, i, j) \otimes \beta(B, k_0, k_m) \otimes \bigotimes_{\substack{1 \leq l \leq n \\ l \neq a}} \bigotimes_{C \in \Lambda} \alpha(C, k_{l-1}, k_l) \\
&= \bigoplus_{m=2}^{n-j+i} \bigoplus_{k_0=0}^{i-1} \bigoplus_{k_m=j}^{n} \bigoplus_{k_0 < \cdots < k_a < k_{a+1} < \cdots < k_m} \\
&\qquad \bigoplus_{B \in \Lambda} \omega(B, i, j) \otimes \beta(B, k_0, k_m) \otimes \bigotimes_{\substack{1 \leq l \leq n \\ l \neq a}} \bigotimes_{C \in \Lambda} \alpha(C, k_{l-1}, k_l) \\
&= \bigoplus_{m=2}^{n-j+i} \bigoplus_{k_0=0}^{i-1} \bigoplus_{k_m=j}^{n} \bigoplus_{k_0 < \cdots < k_a < k_{a+1} < \cdots < k_m} \sigma'(k_0, k_m) \otimes \bigotimes_{\substack{1 \leq l \leq n \\ l \neq a}} \sigma(k_{l-1}, k_l).
\end{aligned}$$

The above two recursions show that this approach is challenging, despite its elegance. Enumerating the sets of partitions is in itself already a nontrivial task.

4.6.5. Pruned parse forest

Another solution is to prune the parse forest: to keep only those trees that can be obtained by the normal form transformation. Going back to the example at the beginning of the example, this solution asks us to assign probability zero to the right tree, the tree that we cannot obtain by the normal form transformation. Since the transformation that we apply is very regular, we can easily characterize the set of trees that are impossible to derive. In fact, it boils down to one characteristic: a node can have the label $\varnothing$ only

when *either* that node spans a single word *or* that node is the right child of its parent node.[8] That means we should remove from the parse forest all edges of the form illustrated in figure 4.7. By this same reasoning we conclude that the parse forest should also be rid of all nodes of the form $(\varnothing, 0, j)$ for $j > 1$, which cannot occur in any tree obtained with our transformation.

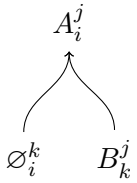


Figure 4.7.: Edges in the parse forest that produce the derivational ambiguity, whenever $k > i + 1$.

With this characterization in hand, we can alter the recursions, starting with the inside algorithm.

$$\begin{aligned}
\alpha(A,i,j) &= \bigoplus_{B\in\Lambda}\bigoplus_{C\in\Lambda} \omega(A,i,j)\otimes\alpha(B,i,i+1)\otimes\alpha(C,i+1,j) \\
&\quad \oplus \bigoplus_{B\in\Lambda\setminus\{\varnothing\}}\bigoplus_{C\in\Lambda}\bigoplus_{k=i+2}^{j-1} \omega(A,i,j)\otimes\alpha(B,i,k)\otimes\alpha(C,k,j) \\
&= \omega(A,i,j)\otimes\Bigg[\bigoplus_{B\in\Lambda}\alpha(B,i,i+1)\otimes\bigoplus_{C\in\Lambda}\alpha(C,i+1,j) \\
&\quad \oplus \bigoplus_{k=i+2}^{j-1}\bigoplus_{B\in\Lambda\setminus\{\varnothing\}}\alpha(B,i,k)\otimes\bigoplus_{C\in\Lambda}\alpha(C,k,j)\Bigg] \\
&= \omega(A,i,j)\otimes\Bigg[\sigma(i,i+1)\otimes\sigma(i+1,j)\oplus\bigoplus_{k=i+2}^{j-1}\tilde{\sigma}(i,k)\otimes\sigma(k,j)\Bigg]
\end{aligned}$$

where

$$\tilde{\sigma}(i,k) = \bigoplus_{B\in\Lambda\setminus\{\varnothing\}} \alpha(A,i,k).$$

For the outside algorithm we need to split cases: the outside algorithm sums over ways to complete a labeled span into a larger labeled span, and as we have noted, the node (A, i, j), for all $A \neq \varnothing$, is allowed to combine in ways that $(\varnothing, i, j)$ is not, whenever $j > i + 1$. Specifically, when $j > i + 1$, the node $(\varnothing, i, j)$ is not allowed to combine *with any node* of the form (C, j, k) for any $C \in \Lambda$. That would constitute an edge that we

[8] We binarize *rightwards*, thus introducting the dummy labels in that position, and can only ever introduce the dummy label in the left child position when it spans a single word.

deemed illegal. We thus split the computation of $\beta(A, i, j)$ into the following two cases: the case where $j > i + 1$ and $A = \varnothing$, and otherwise.

First, consider the case where either $A \neq \varnothing$, or $j = i+1$. In this case, the node (A, i, j) can combine any way it desires, as the left node, and as the right node, but note that when we complete it as the right node—thus adding a node to it on the left—we still need to avoid completing it with $(\varnothing, k, i)$ if $k \neq i - 1$. Taking this into account we can derive

$$\begin{aligned}
\beta(A, i, j) = &\bigoplus_{B \in \Lambda} \bigoplus_{C \in \Lambda} \omega(B, i-1, j) \otimes \alpha(C, i-1, i) \otimes \beta(B, i-1, j) \\
&\oplus \bigoplus_{B \in \Lambda} \bigoplus_{C \in \Lambda \setminus \{\varnothing\}} \bigoplus_{k=1}^{i-2} \omega(B, k, j) \otimes \alpha(C, k, i) \otimes \beta(B, k, j) \\
&\oplus \bigoplus_{B \in \Lambda} \bigoplus_{C \in \Lambda} \bigoplus_{k=j+1}^{n} \omega(B, i, k) \otimes \beta(B, i, k) \otimes \alpha(C, j, k) \\
= &\sigma'(i-1, j) \otimes \sigma(i-1, i) \oplus \bigoplus_{k=1}^{i-2} \sigma'(k, j) \otimes \tilde{\sigma}(k, i) \oplus \bigoplus_{k=j+1}^{n} \sigma'(i, k) \otimes \sigma(j, k).
\end{aligned}$$

Now consider the special case where $A = \varnothing$ and $j > i + 1$. In this case we must disallow the node to combine in the left position. This corresponds simply to removing that part of the sum; in the above derivation this is the third term. This gives:

$$\beta(A, i, j) = \sigma'(i-1, j) \otimes \sigma(i-1, i) \oplus \bigoplus_{k=1}^{i-2} \sigma'(k, j) \otimes \tilde{\sigma}(k, i)$$

4.7. Related work

The model that we presented regards a constituency tree as a collection of *labeled spans* over a sentence. Earlier CRF models for constituency parsing, both log-linear and neural, factorize trees over *anchored rules* [Finkel et al., 2008, Durrett and Klein, 2015]. This puts most of the expressiveness of the model in the state space of the dynamic program—modelling interactions between subparts of the trees through their interaction in the rules—instead of at the feature level. The model in Stern et al. [2017a] removes part of this structure, and puts more expressiveness in the input space by using rich neural feature representations conditioning on the entire sentence. The discrete interaction between the local scores remains only at level of labeled spans. This dramatically improves the speed of this model, which will become evident in the next section.

In recent decades, discriminative chart-parsing has moved from grammar to features. Hall et al. [2014] show how the log-linear CRF model of Finkel et al. [2008] can work with bare unnanotated grammars when relying more heavily on surface features of the sentence. Durrett and Klein [2015] show how the linear scoring function in the

model of Hall et al. [2014] can be replaced by a neural network. The work of [Stern et al., 2017a] moves one step further: the model is span-factored—thus dispensing with the structure of a grammar altogether—and the scoring function can condition on the surface features of the entire sentence with the use of recurrent neural networks.

Contrast this with generative parsing based on treebank grammars, where features are not available because the models are not conditional. These models instead rely entirely on detailed rule information. Basic treebank grammars do not parse well because the rules provide too little context, and good results can only be obtained by enriching grammars. The independence assumptions in the grammar are thus typically weakened, not strengtehened. Such approaches lexicalize rules [Collins, 2003], annotate the rule with parent and sibling labels [Klein and Manning, 2003], or automatically learn refinements of nonterminal categories [Petrov et al., 2006].

In terms of the probabilistic model, the closest predecessor to our model is the neural CRF parser of Durrett and Klein [2015], which predicts local potential for anchored rules using a feedfoward network. It differs from our approach in two ways. Their method requires a grammar, extracted from a treebank beforehand, whereas our approach implictly assumes all rules are possible rules in the grammar. Secondly, their scoring function conditions only on the parts of the sentence directly under the rule, dictated by the use of a feedforward network, whereas our scoring function computes a score bassed on representations computed from the entire sequence.

5. Syntactic evaluation

Language models are typically evaluated by the perplexity they assign to held out data, and thus did we evaluate our language models in the previous chapters. In this chapter we look at alternatives. In particular, we look at evaluation that specifically probes the syntactic abilities of a language model. To this end we take the dataset introduced by Marvin and Linzen [2018], which presents a comprehensive set of syntactic challenges, and evaluate the models presented thus far against them. The dataset consists of constructed sentence pairs that differ in only one word, making one sentence grammatical and the other not. The task is to assign higher probability to the grammatical sentence. We can think of this task as soliciting comparative *acceptability judgements*—a key concept in linguistics. We will refer to this dataset as Syneval, for *syntactic evaluation*.

In this chapter:

- We introduce the Syneval dataset: we describe syntactic phenomena that it tests and review how this dataset relates to other work on syntactic evaluation.
- We introduce multitask learning with a syntactic side objective as an previously proposed approach to improve language models for this task: we describe a previous approach that combines language modelling with CCG supertagging [Enguehard et al., 2017], and introduce a novel multitask language model based on span labeling that is inspired by the scoring function of our CRF parser.
- We evaluate all the models introduced thus far on this dataset and show that this gives finegrained differentation between the various models, generally favoring the RNNG, but often closely followed by and sotemies surpassed by the multitask models.

5.1. Syntactic evaluation

A shortcoming of perplexity is that the metric conflates various sources of success [Marvin and Linzen, 2018]. A language model can make use of many aspects of language to predict the probability of a sequence of words. And although we would like the probability of a sentence to depend on high-level phenomena such as syntactic well-formedness or global semantic coherence, a language model can also focus on lower hanging fruit such as collocations and other semantic relations predicted from local context. Syntax is especially difficult to evaluate given that most sentences in a corpus are grammatically simple [Marvin and Linzen, 2018]. Arguably, this conflation is also the appeal: perplexity is a one size fits all metric. But for a fine-grained analysis we must resort to a fine-grained metric.

Recently, a series of papers has introduced tasks that specifically evaluate the syntactic abilities of language models [Linzen et al., 2016, Gulordava et al., 2018, Marvin and Linzen, 2018]. And such tasks can be very revealing. The task introduced by Linzen et al. [2016] is to predict the correct conjugation of a verb given an earlier occuring subject—especially in the presence of distracting subjects that intervene[1]. This task has revealed that lower perplexity does not imply greater success on this task [Tran et al., 2018], and that an explicitly syntactic model like the RNNG significanlty outperforms purely sequential models, especially with increasing distance between the subject and the verb [Kuncoro et al., 2018]. This type of agreement is one of the phenomena evaluated in Syneval.

5.1.1. Dataset

The Syneval dataset consists of contrastive sentence pairs that differ in only one word. Let (x, x') be this minimal pair, with grammatical sentence x and an ungrammatical sentence x'. Then a language model p makes a correct prediction on this pair if $p(x) > p(x')$.

The classification is based on the probability of the entire sequence. This makes the task applicable to grammatical phenomena that involve interaction between multiple words, or where the word of contention does not have any left context. This contrasts with the task introduced in Linzen et al. [2016]. This approach is also more natural for a model like the RNNG, where the probability of the sentence is computed by marginalizing over all latent structures, whereas individual word probabilities can only be obtained when conditioning on a single structure—for example a predicted parse—as is done in Kuncoro et al. [2018].

The sentence pairs fall into three categories that linguists consider to depend crucially on hierachical syntactic structure [Everaert et al., 2015, Xiang et al., 2009]:

1. Subject-verb agreement (The farmer *smiles*.)
2. Reflexive anaphora (The senators embarassed *themselves*.)
3. Negative polarity items (*No* authors have *ever* been famous.)

The dataset contains constructions of increasing difficulty for each of these categories. For example, the distance between two words in a syntactic dependency can be increased by separating them with a prepositional phrase: *The farmer next to the guards smiles*. In this example *the guards* additionally forms a distractor for the proper conjugation of *smiles*, making the example extra challenging. The dataset is constructed automatically using handcrafted context-free grammars. The lexical rules are finegrained so that the resulting sentences are semantically plausible. In particular there are rules for animate and innanimate objects so a sentence like *The apple laughs* cannot be constructed. The total dataset consists of around 350,000 sentence pairs. The full set of sentence types per category, including examples, is listed appendix D.

[1] E.g. *Parts* of the river valley *have/has*.

5.1.2. RNNG

What would be the advantage of the RNNG on this task? To illustrate this, consider the following the example taken from the dataset, which will also illustrate the important role played by the posterior. One of the classes—long VP coordination—has the sentence pair

(1) a. the author knows many different foreign languages and *likes* to watch television shows
b. *the author knows many different foreign languages and *like* to watch television shows

as first example. Parsing the grammatical sentence with the discriminative RNNG gives the sensible analysis

(2) [$_{S}$ [$_{NP}$ the author] [$_{VP}$ [$_{VP}$ knows [$_{NP}$ many different foreign languages] and [$_{VP}$ likes [$_{S}$ [$_{VP}$ to [$_{VP}$ watch [$_{NP}$ television shows]]]]]]]

as prediction, while parsing the ungramatical sentence results in

(3) [$_{S}$ [$_{NP}$ the author] [$_{VP}$ knows [$_{NP}$ many different foreign languages] [$_{FRAG}$ and like [$_{S}$ [$_{VP}$ to [$_{VP}$ watch [$_{NP}$ television shows]]]]]]].

The first analysis correctly recognizes the verb phrase 'knows x and likes y' as the conjunction of two verb phrases. The most likely analysis of the odd *like* in the second sentence seems to be dangle as a fragment. For the grammatical tree the samples all correspond closely to the predicted parse, while the samples for the second sentences vary greatly, containing such unlikely trees as

(4) [$_{S}$ [$_{NP}$ the author] [$_{VP}$ knows [$_{NP}$ many different foreign languages] and [$_{PP}$ like [$_{S}$ [$_{VP}$ to [$_{VP}$ watch [$_{NP}$ television shows]]]]]]

that interpret *like* as a preposition. A similar pattern is observed for the CRF parser, in which case the observed difference in the conditional distributions can additionally be confirmed by their difference in entropy: 1.43 for the grammatical sentence against 4.23 for the ungrammatical one. In other words: for the grammatical sentence the generative RNNG will receive quality trees of high likelihood, while for the ungrammatical sentence incoherent trees of low likelihood. Of course, the final probability of the sentence will depend equally on the joint probability that the RNNG will asign to the trees.

5.2. Multitask learning

One method that makes language models perform better in the tasks described in this chapter is to provide stronger syntactic supervision by using multitask learning [Enguehard et al., 2017, Marvin and Linzen, 2018]. Multitask learning is a simple yet effective way of providing additional supervision to neural network models by combining

multiple tasks in a single objective while sharing parameters [Caruana, 1997], and has been succesfully applied in natural language processing [Collobert and Weston, 2008, Collobert et al., 2011, Zhang and Weiss, 2016, Søgaard and Goldberg, 2016]. By combining objectives that share parameters the model is encouraged to learn representations that are useful in all the tasks.

In this section we describe two simple baselines for the syntactic evaluation task that are based on multitask learning. Both methods are based on language modelling with a syntactic side objective. The first side objective is to predict combinatory categorical grammar (CCG) supertags [Bangalore and Joshi, 1999] for each word in the sentence, and is proposed in [Enguehard et al., 2017]. The second side objective is to label spans of words with their category. This objective is inspired by the label scoring function in the CRF parser introduced in chapter 4.[2] Both objectives are challenging and should require representations that encode a fair amount of syntactic information.

5.2.1. Multitask objective

In our case we combine a language model p with a syntactic model q, and optimize these jointly over a single labeled dataset $\mathcal{D}$[3]. In this case, we maximize the objective

$$\mathcal{L}(\theta, \lambda, \xi) = \sum_{(x,y)\in\mathcal{D}} \log p_{\theta,\lambda}(x) + \log q_{\theta,\xi}(y|x) \tag{5.1}$$

with respect to the parameters θ, λ and ξ. The key feature of multitask learning is that the two models p and q share the set of parameters θ and that in objective 5.1 these parameters will thus be optimized to fit *both* the models well. The parameters in λ and ξ are optimized for their respective objectives separately. The proportion and the nature of the parameters that belong to θ is a choice of the modeller and determines how both objectives influence one another. In the following paragraphs we specify this parametrization.

5.2.2. Models

We have introduced the multitask model as consisting of 2 models, p and q. The main model p is a regular RNN language model on sentences x:

$$\log p_{\theta,\lambda}(x) = \sum_{i=1}^{n} \log p_{\theta,\lambda}(x_i \mid x_{<i}),$$

[2] This is similar to an approach found in recent work on semantic parsing where a similar side-objective is used and where it is called a 'syntactic scaffold' [Swayamdipta et al., 2018]. The exact form of our side-objective is novel, as far as the author is aware of, and is parametrized in a considerably simpler way.

[3] Note that it is our choice to focus on only a single dataset, and that multitask learning is principle more flexible than that, providing the option to combine multiple disjoint datasets in a single objective.

where the conditional probabilities of x_i are computed by linear regression on forward RNN vectors $\mathbf{f}_{i-1}$ as

$$p_{\theta,\lambda}(x \mid x_{<i}) \propto \exp\Big\{[\mathbf{W}\mathbf{f}_{i-1} + \mathbf{b}]_x\Big\},$$

where x doubles as an index. The parameters $\lambda = \{\mathbf{W}, \mathbf{b}\}$ are specific to the language model, and the vectors $\mathbf{f}_i$ are computed using a forward RNN parametrized by θ. The vectors $\mathbf{f}_i$ are used in the side objective as well, and it is in this precise sense that the parameters θ are shared between p and q.

Word labeling Let $y = \langle y_1, \ldots, y_n \rangle$ be a sequence of CCG supertags for the sentence x, with one tag for each word. The side model is then a simple greedy tagging model:

$$\log q_{\theta,\xi}(y \mid x) = \sum_{i=1}^{n} \log q_{\theta,\xi}(y_i \mid x).$$

The probability over tags for position i are computed from $\mathbf{f}_i$ using a feedforward network parametrized by ξ

$$q_{\theta,\xi}(y_i \mid x) \propto \exp\Big\{[\textsc{Ffn}_\xi(\mathbf{f}_i)]_{y_i}\Big\},$$

where y_i doubles as index. This side objective is taken from Enguehard et al. [2017] and is the side objective that is used in the multitask model of Marvin and Linzen [2018].

Span labeling Let y be a sequence of labeled spans (A_k, i_k, j_k) obtained from a gold parse tree for x. Given span endpoints i and j and the sentence x, the side model q predicts a label A:

$$\log q_{\theta,\xi}(y \mid x) = \sum_{k=1}^{K} \log q_{\theta,\xi}(A_k \mid x, i_k, j_k).$$

The representation for the span is defined as in the CRF parser as

$$\mathbf{s}_{ij} = \mathbf{f}_j - \mathbf{f}_i,$$

and the distribution over labels is computed using a feedforward network parametrized by ξ as

$$q_{\theta,\xi}(A \mid x, i, j) \propto \exp\Big\{[\textsc{Ffn}_\xi(\mathbf{s}_{ij})]_A\Big\},$$

where A doubles as index. This model differs from the supertagging model in the interaction of the vectors $\mathbf{f}$ in the definition of $\mathbf{s}_{ij}$. This linear defintion puts significant restraints on the encodings, which is precisely what we want. This side objective is inspired by the label scoring function of the CRF parser, and is novel as a multitask objective, as far as we know.

	RNNLM	RMMLM-span	RNNLM-CCG
ppl	$97.77 \pm 0.65 \quad (97.32)$	$102.92 \pm 0.64 \quad (102.39)$	$143.9 \pm 1.79 \quad (143.04)$

Table 5.1.: Perplexity of the language models on the PTB test set.

5.3. Experiments

In this section we report the results of the evaluation on the Syneval dataset. We evaluate all the models proposed thus far: the generative RNNG with the two proposal models as described in chapters 3 and 4; the multitask models described in the previous section; and a baseline RNN language model. We first describe the training setup and results of the (multitask) RNN language models. We then describe the evaluation setup and discuss the results.

5.3.1. Setup

We first train the three language models. We train one baseline RNN language model (RNNLM), and two RNN language models trained with multitask learning using the span side objective (RNNLM-span) and the CCG side objective (RNNLM-CCG). Because the RNNLM will function as a baseline comparison to the RNNG we choose the dimensions of the states to result in a similar number of parameters. The models use the same vocabulary as the generative RNNG, and the embeddings are of size 100 as well. The LSTM has dimension 330 and has 2 layers. The feedforward networks used by the side models q in the multitask models have 1 hidden layer with dimension 250. The number of parameters the models totals around 9.3M parameters, not counting the parameters of q which we discard after training. We train the three language models exactly as the generative RNNG, using SGD with a learning rate of 0.1, and dropout of 0.3.

We train 10 separate seperate runs of each model. The perplexity results are shown in table 5.1. The multitask models take considerably more epochs to converge[4] and perform worse in terms of perplexity than the single task language model. The gap with the RNNLM-CCG model is particularly large. This contrasts with Marvin and Linzen [2018] who obtain a lower perplexity with CCG tagging than without. This opposite result is probably caused by a difference in training data. Marvin and Linzen [2018] use an unlabeled dataset of 90 million words—almost 100 times the size of the PTB—which puts much more weight on the language modelling part of the multitask objective, favoring the word prediction over the tagging.

5.3.2. Results

We evaluate four models on the syneval dataset: the generative RNNG with RNNG proposals (RNNG) and CRF proposals (RNNG-CRF); the mulitask RNNLM models

[4] Around 150 versus 100.

(RNNLM-span and RNNLM-CCG); and the RNNLM baseline. For the RNNG we use 100 samples and for the RNNG proposals we use a temperature of 0.8. We evaluate each of the 10 trained runs per model and report the mean and standard deviation on the dataset. We do not preprocess the Syneval dataset in any way, leaving the sentences with their original lowercase initial letter and without period at the end, and keep all sentences that contain unknown words. The results are shown in figure 5.1 broken down per detailed category, corresponding to the list in appendix D. The results with a less finegrained view can bee seen in figure 5.2, showing the average accuracy per broad category.[5]

In 5.1 we can see that overall the results are close, especially taking into considering the deviation within each model. The most striking immediate fact is that the accuracy in the NPI category shows enormous variation, while on average performing rather poorly. This variation is *between* models, but most importantly it is *within* the runs of single models. The variation within model is so great for this class that we think it precludes us from drawing many conclusions about it other than these: that none of the models converge to a consistent attitude with respect to negative polarity items; and that what some of these models learn seems te be quite the *opposite* of what is expected. In categories 1 and 2, things are looking more definitive within models, but less definitive between models. One category stands out: for the object relative clauses the RNNG outperforms the other models by a large margin. The results in the other categories look less convincing.

When we move to figure 5.1 and consider the accuracies on average we see a clearer pattern: the RNNG performs best in the first category, and the multitask models perfom best in the second. But the results remain rather close. The RNNG also outperforms the others in the third category—on average, because again we can see the large deviation within the different runs.

We also investigate the impact of some basic preprocessing of the Syneval dataset. Uppercasing the initial letter of each sentence resulted in much lower perplexities for each sentence—to be expected because we use case sensitive vocabularies—but did not affect the classification accuracy in any category, indicating that capitalization does not affect the comparative differences in perplexity. Removing all pairs with unknown words—about half of the dataset—did affect the classification, but not always in a clear way. For the RNNG and RNNLM we did find an interesting pattern, shown in figures 5.3 and 5.4. For both models the results are better without unknown words, but the impact on the RNNLM is much greater. In particular, in the first three subcategories of the agreement category, the RNNLM is worse than the RNNG when including unknown words, but outperforms the RNNG when the unknown words are excluded. This shows that in the ideal situation the RNNLM can learn agreement as well as the RNNG, but that it has a hard time dealing with the grammatical function of the unknown words compared to the RNNG. Quite possibly this shows the added benefit of

[5]This average is computed over the *accuracies* per subcategory in figure 5.1 not over the per-sentence *predictions*, thus disregarding the difference in number of sentences per detailed category. The standard deviation is still over the runs.

structural information in the interpretation of the unknown words.

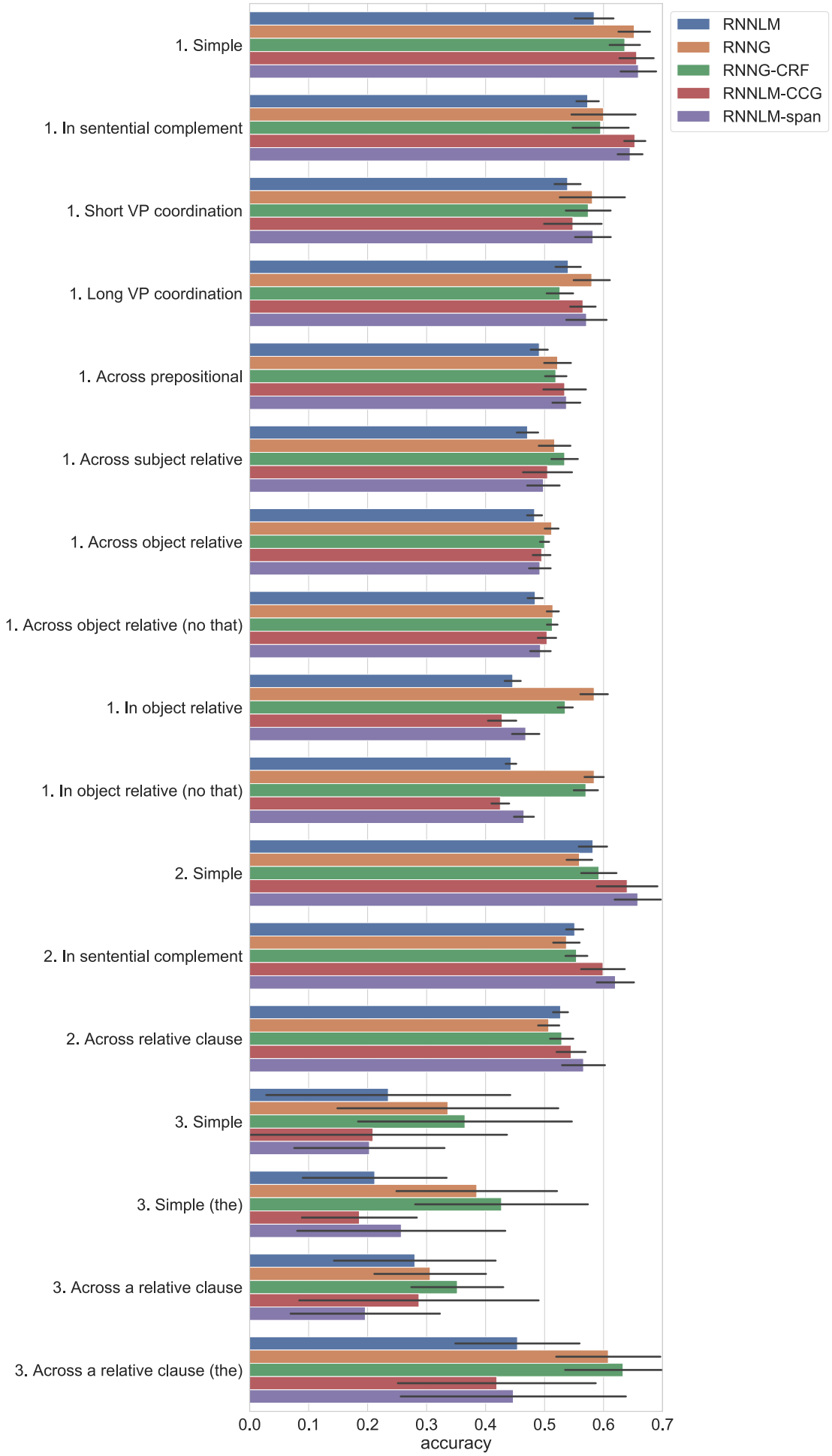


Figure 5.1.: Syneval results for all models per category, bars showing standard deviation. The categories are described in appendix D.

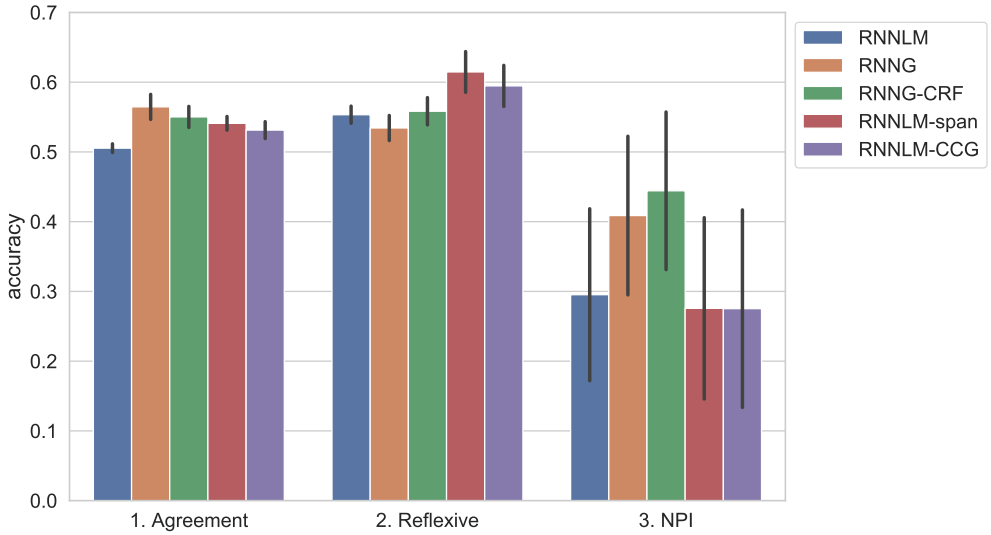


Figure 5.2.: Syneval results averaged over the three main categories.

5.4. Related work

There has been a surge recently of work on syntactic evaluation. Linzen et al. [2016] introduce the task of long distance subject-verb agreement and Gulordava et al. [2018] make this test more challenging by turning the sentences nonsensical while keeping them grammatical. Both datasets are extracted from a wikipedia corpus based on properties of their predicted dependency parse. To make the sentences nonsensical, Gulordava et al. [2018] randomly substitute words from the same grammatical category.[6] Warstadt et al. [2018] fine-tune neural models to learn to immitate grammatical acceptability judgments gathered from linguistics textbooks.

McCoy et al. [2018] train a neural machine translation that turns a declarative sentence into a question, a kind of transformation that linguists have argued requires the existence of hierarchical structure in language [Everaert et al., 2015].[7]

Finally, targeted evaluation has been introduced previously for semantic comprehension by Zweig and Burges [2011] in a sentence completion task, and minimal contrastive sentence pairs have been used to evaluate neural machine translation by Sennrich [2017].

[6] An approach inspired by Chomsky's (in)famous sentence *Colorless green ideas sleep furiously* that is both grammatical and nonsensical.

[7] Such declarative-question pairs play a central role as empirical evidence in the argument—known as the *argument from the poverty of the stimulus*—that humans have an innate predisposition for generalizations that rely on hierarchical structure rather than linear order [Chomsky, 1980].

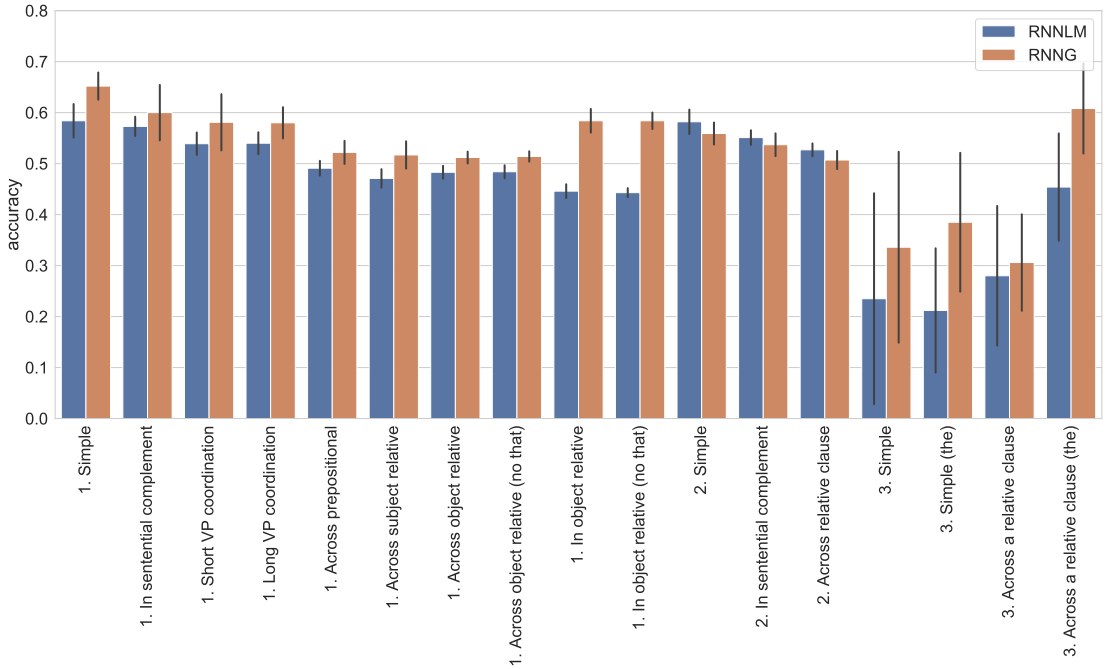


Figure 5.3.: Syneval results for RNNLM and RNNG.

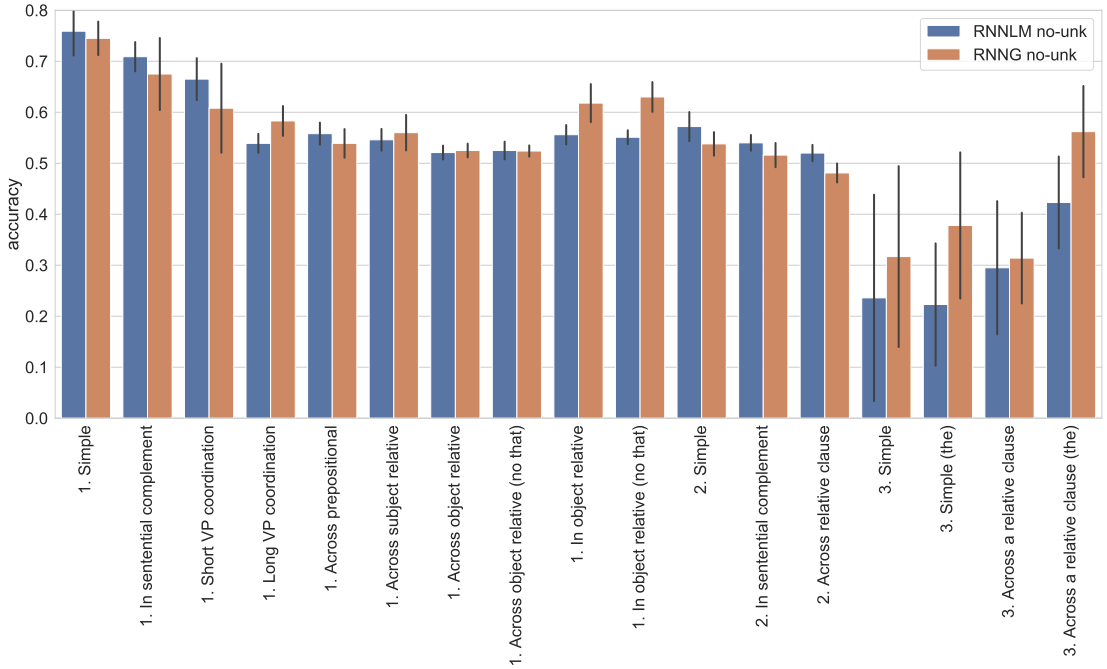


Figure 5.4.: Syneval results for RNNLM and RNNG, pairs with unknown words removed.

6. Semisupervised learning

The generative RNNG defines a joint distribution over trees and sentences. In chapter 3 we showed how this distribution can be estimated from labeled data. In this chapter we show how estimation can be extended to unlabeled data. We optimize a tractable lower bound on the marginal log likelihood using amortized variational inference with an approximate posterior—a procedure reminiscent of the importance sampling inference, and both the discriminative RNNG as the CRF can be used. To compute gradients we use samples from the posterior, and the CRF particularly excels in this role: the entropy term in the lower bound can be computed exactly, and the global normalization of the distribution makes the model a well behaved sampler. The lower bound allows us to formulate a semisupervised training objective. This in principle allows us to add any amount of unlabeled data to the already existing labeled data, thus extending the training to both more data and different domains. A further question is whether the RNNG can be estimated from unlabeled data alone. This is particularly challenging because the RNNG makes no independence assumptions and as such it is questionable whether a model so expressive possesses enough inductive bias to induce any consistent structure without supervision. Very recently [Kim et al., 2019] have shown the success of this approach for unlabeled binary trees in an approach that is remarkably similar to ours, also making use of a CRF inference model.

[...]

In this chapter:

- We describe how the RNNG can be used to learn from unlabeled data using amortized variational inference with an approximate posterior.
- We show how to obtain gradients for of the lowerbound by using the score function estimator, and show how to reduce the variance of this estimator using baselines.
- We describe how the discriminative RNNG and the CRF parser introduced in chapter 4 both can be used as approximate posterior, but emphasize how a globally normalized CRF excels in this role.
- We describe experiments with semisupervised and unsupervised training—with both the CRF and RNNG as posteriors, for both labeled an unlabeled trees, with and without supervised pretraining—and obtain no definitive results, but analyze our preliminary findings.
- We conclude that with the appropriate changes to the CRF posterior, the unsupervised and unlabeled learning of the RNNG could prove succesful, indicated

by the very recent success of Kim et al. [2019] with this approach for unlabeled binary trees.

6.1. Unsupervised learning

We first describe how the joint distribution of the RNNG can be estimated from unlabeled data by maximizing the marginal likelihood, making the trees a latent variable. We derive the lower bound and describe how the choice of approximate posterior affects the optimization of it.

6.1.1. Variational approximation

The joint log likelihood $\log p_\theta(x, y)$ of the generative RNNG defines a marginal likelihood

$$\log p_\theta(x) = \log \sum_{y \in \mathcal{Y}(x)} p_\theta(x, y).$$

Optimizing this with respect to θ directly is intractable due to the sum over trees and so we must resort to an approximate method. We use variational inference [Jordan et al., 1999, Blei et al., 2016] and introduce a posterior $q_\lambda(y|x)$ parametrised by λ and use Jensen's inequality to derive a variational lower bound on the marginal likelihood:

$$\begin{aligned}
\log p(x) &= \log \sum_{y \in \mathcal{Y}(x)} q_\lambda(y|x) \frac{p_\theta(x, y)}{q_\lambda(y|x)} \\
&= \log \mathbb{E}_{q_\lambda} \left[\frac{p_\theta(x, y)}{q_\lambda(y|x)} \right] \\
&\geq \mathbb{E}_{q_\lambda} \left[\log \frac{p_\theta(x, y)}{q_\lambda(y|x)} \right]. \\
&= \mathbb{E}_{q_\lambda}[\log p_\theta(x, y) - \log q_\lambda(y|x)]. \qquad (6.1)
\end{aligned}$$

This is called the evidence lower bound (ELBO) [Blei et al., 2016], and we define it as a function of parameters θ and λ given a single datapoint x as

$$\mathcal{E}(\theta, \lambda; x) = \mathbb{E}_{q_\lambda} \left[\log \frac{p_\theta(x, y)}{q_\lambda(y|x)} \right]. \qquad (6.2)$$

The objective is optimized with respect to both the generative and inference parameters. We can rewrite it in two ways, each providing different perspective on this optimization procedure. On the one hand we can write the as [Blei et al., 2016]

$$\mathcal{E}(\theta, \lambda; x) = \log p_\theta(x) - \mathrm{KL}(p_\theta(y \mid x) || q_\lambda(y \mid x)).$$

This reveals that maximizing the ELBO is equivalent to minimizing the KL divergenge between the approximate posterior $q_\lambda(y \mid x)$ and the true posterior $p_\theta(y \mid x)$, while

maximizing the marginal log likelihood $\log p_\theta(x)$. Because the true posterior $p_\theta(y \mid x)$ cannot be computed efficiently—for the same reason that we cannot compute the marginalization—this formulation has no purposes for us other than theoretical insight. The alternative formulation will have practical purpose:

$$\begin{aligned}\mathcal{E}(\theta, \lambda; x) &= \mathbb{E}_{q_\lambda}[\log p_\theta(x, y)] - \mathbb{E}_{q_\lambda}[\log q_\lambda(y|x)] \\ &= \mathbb{E}_{q_\lambda}[\log p_\theta(x, y)] + H_{q_\lambda}(Y|X = x). \end{aligned} \tag{6.3}$$

The first term is the expectation of the joint model under the the posterior distribution, and the second is the entropy of that distribution. This reveals that the objective is twofold: the posterior is encouraged to put its mass on those trees that have likelihood under the joint model, while the entropy regularizes q_λ from overly concentrating probability mass. The first part of the objective is now in the form of an expectation which we can approximate using samples from the our approximate posterior. Whether the entropy can be computed depends on the choice of posterior, and can otherwise be estimated with samples as well.

6.1.2. Approximate posterior

As approximate posterior we can use both the RNNG and CRF: both models satisfy the condition that their support is a subset of the support of the true posterior $p(y \mid x)$, which is required for the ELBO optimization [Kucukelbir et al., 2017]. The support of the RNNGs match up. The support of the CRF is a strict subset, because although it can handle common unary chains, it cannot handle the arbitrary chains that are in the support of the generative RNNG. Because this is a marginal phenomenon the space outside the reach of the CRF will be low density anyhow.

6.2. Training

We use the ELBO to formulate a semisupervised and an unsupervised optimization objective. Let $\mathcal{D}_L = \{(x_i, y^{(k)})\}_{i=1}^N$ be the familiar dataset from the supervised training, and let $\mathcal{D}_U = \{x_i\}_{i=1}^M$ be an unlabeled dataset consisting merely of sentences x. The unsupervised objective is then to maximize

$$\mathcal{E}(\theta, \lambda) = \sum_{x \in \mathcal{D}_U} \mathcal{E}(\theta, \lambda; x), \tag{6.4}$$

while the semisupervised objective combines the supervised and the unsupervised objective into one as

$$\mathcal{L}(\theta, \lambda) = \mathcal{L}_S(\theta) + \mathcal{E}(\theta, \lambda),$$

where $\mathcal{L}_S(\theta) = \sum_{(x,y)\in\mathcal{D}} \log p_\theta(x, y)$ is the supervised objective.

The objectives are optimized with gradient based optimization, which means that we need to compute the gradients $\nabla_\theta \mathcal{E}(\theta, \lambda)$ and $\nabla_\lambda \mathcal{E}(\theta, \lambda)$. These gradients are estimated

with samples from the posterior, and the form of those estimates will depend on the posterior model used. With the CRF as posterior, we write ELBO as

$$\mathcal{E}_{\text{CRF}}(\theta, \lambda; x) = \mathbb{E}_{q_\lambda}[\log p_\theta(x, y)] + H_{q_\lambda}(Y|X = x), \tag{6.5}$$

to emphasize that we can compute the entropy exactly and that only the first part of the sum needs to be estimated. With the RNNG as posterior we write the ELBO as

$$\mathcal{E}_{\text{RNNG}}(\theta, \lambda; x) = \mathbb{E}_{q_\lambda}[\log p_\theta(x, y) - \log q_\lambda(y|x)], \tag{6.6}$$

emphasizing that the entire quantity needs to be estimated. We stress however, that this is a purely practical distinction for the objectives are analytically identical.

6.2.1. Gradients of generative model

Computing the gradient with respect to θ is easy and permits a straightforward Monte-Carlo estimate:

$$\begin{aligned}
\nabla_\theta \mathcal{E}_{\text{CRF}}(\theta, \lambda; x) &= \nabla_\theta \mathbb{E}_{q_\lambda}[\log p_\theta(x, y)] + \nabla_\theta H_{q_\lambda}(Y|X = x) \\
&= \mathbb{E}_{q_\lambda}[\nabla_\theta \log p_\theta(x, y)] \\
&\approx \frac{1}{K} \sum_{k=1}^{K} \nabla_\theta \log p_\theta(x, y^{(k)})
\end{aligned} \tag{6.7}$$

where $y^{(k)}$ are independent samples from the approximate posterior $q_\lambda(\cdot|x)$. We can move the gradient inside the expectation because q does not depend on θ and for the same reason the gradient of the entropy is zero. In the case of the RNNG posterior we end up with the same estimator:

$$\begin{aligned}
\nabla_\theta \mathcal{E}_{\text{RNNG}}(\theta, \lambda; x) &= \nabla_\theta \mathbb{E}_{q_\lambda}[\log p_\theta(x, y)] - \log q_\lambda(y|x)] \\
&= \mathbb{E}_{q_\lambda}[\nabla_\theta \log p_\theta(x, y)] \\
&\approx \frac{1}{K} \sum_{k=1}^{K} \nabla_\theta \log p_\theta(x, y^{(k)})
\end{aligned} \tag{6.8}$$

6.2.2. Gradients of inference model

Computing the gradient with respect to λ is less straightforward. The difference will be in the expectation: where in the previous derivations we could freely exchange gradients and expectations now we cannot. This requires us to rewrite the gradient as the *score function estimator* [Fu, 2006]. We will first derive the estimator for the RNNG ELBO and define the *learning signal* as everything that is inside the expectation

$$L(x, y) = \log p_\theta(x, y) - \log q_\lambda(y|x). \tag{6.9}$$

We then use equality C.1 to derive

$$\begin{aligned}\nabla_\lambda \mathcal{E}_{\text{RNNG}}(\theta, \lambda) &= \nabla_\lambda \mathbb{E}_{q_\lambda}[L(x, y)] \\ &= \mathbb{E}_{q_\lambda}[L(x, y)\nabla_\lambda \log q_\lambda(y|x)]\end{aligned} \tag{6.10}$$

In this rewritten form the gradient is an expectation, which does permit a straightforward MC estimate:

$$\mathbb{E}_{q_\lambda}[L(x, y)\nabla_\lambda \log q_\lambda(y|x)] \approx \frac{1}{K}\sum_{k=1}^{K} L(x, y^{(k)})\nabla_\lambda \log q_\lambda(x|y^{(k)}) \tag{6.11}$$

where again $y^{(k)}$ are independent samples from $q_\lambda(\cdot|x)$.

For the CRF posterior we need the score function estimator only for the part inside the expectation—the gradient of the entropy can be computed exactly. We thus define

$$L(x, y) = \log p_\theta(x, y), \tag{6.12}$$

and derive

$$\begin{aligned}\nabla_\lambda \mathcal{E}_{\text{CRF}}(\theta, \lambda) &= \nabla_\lambda \mathbb{E}_{q_\lambda}[L(x, y)] + \nabla_\lambda H_{q_\lambda}(Y|X = x)] \\ &= \mathbb{E}_{q_\lambda}[L(x, y)\nabla_\lambda \log q_\lambda(y|x)] + \nabla_\lambda H_{q_\lambda}(Y|X = x).\end{aligned} \tag{6.13}$$

The computation of $H_{q_\lambda}(Y|X = x)$ is fully differentiable (*cf.* 4.15), and so we can rely on automatic differentiation to compute $\nabla_\lambda H_{q_\lambda}(Y|X = x)$.

Estimators of this form have been derived by Williams [1992], Paisley et al. [2012], Mnih and Gregor [2014], Ranganath et al. [2014], Mnih and Rezende [2016] and Miao and Blunsom [2016], and is also known as the *reinforce estimator* [Williams, 1992].

6.2.3. Variance reduction

The score function estimator is unbiased but is known to have high variance—often too much to be useful in practice [Paisley et al., 2012]. To reduce variance we use a data dependent baseline $b(x)$ and redefine the estimator as

$$\frac{1}{K}\sum_{k=1}^{K}(L(x, y^{(k)}) - b(x))\nabla_\lambda \log q_\lambda(x|y^{(k)}). \tag{6.14}$$

The more the value $b(x)$ correlates with with $L(x, y^{(k)})$ the greater the reduction in variance, but the baseline cannot depend on the samples $y^{(k)}$ (*cf.* appendix C). We use a clever baseline introduced by Rennie et al. [2017] which is based on the argmax decoding of the posterior, that almost depends on the samples without being a control variate. Letting $\hat{y} = \arg\max_y q_\lambda(y \mid x)$ we define

$$b(x) = L(x, \hat{y}). \tag{6.15}$$

The reasoning is elegant: the samples will tend to look like $\hat{y}$ when the mass of the q_λ concentrates, in turn making $L(x, \hat{y})$ close to $L(x, y^{(k)})$. And while this is a data dependent baseline it involves no additional parameters, in contrast with baselines based on feedforward networks [Mnih and Gregor, 2014, Miao and Blunsom, 2016] or RNN language models [Yin et al., 2018]. For the RNNG, we compute use the gready approximation y^* but with the CRF we can obtain $\hat{y}$ exactly.

6.3. Experiments

The above training objectives give us a range of options to investigate, and we explore them all. We have two posteriors at our disposal, labeled and unlabeled data, and the option to work with unlabeled trees. Yet unfortunately, none of the experiments we describe will lead to any defnitive results. But while we show how some approaches do not work for reasons practical and theoretical, other directions look promising. One stands out: to use the CRF posterior as posterior, on unlabeled trees, in the semisupervised and even unsupervised learning setting. In fact concurrent work by Kim et al. [2019]—recently published and remarkably similar to ours—shows that, with the proper optimization strategies, such an approach can be made to work with unlabeled, binary trees. Our approach differs from theirs because we dot not restrict our CRF to binary trees, and use the original formulation of the RNNG and not a simpler binary TreeLSTM. Unfortunately—as we will describe—the derivational ambiguity in the CRF comes back with a vengeance, and our attempts at this approach strand. For now.

We now describe the experiments that we performed, and the preliminary results that we obtained. We use the same model architectures as before, and details about optimization and data are in appendix A. We finish on a positive and describe the future work that will make it likely for our CRF approach to work.

6.3.1. RNNG posterior

We experimented with the discriminate RNNG posterior in the semisupervised setting, using the One Billion Word benchmark to obtain unlabeled data. Altough we use pretrained models we do not obtain succes. We consistenly find that the model deteriorates to pathological trees, typically producing endless unary chains of the same symbol. This automatically stops the training by causing problems with numerical stability in the action distributions. We decide against further exploration, and focuss on the CRF posterior instead.

6.3.2. CRF posterior

Initial experiments with a pretrained CRF on the semisupervised objective—analog to the RNNG experiment—show us the practical limitations of the CRF posterior: this combination is strikingly slow. We fit not even one epoch in 48 hours, and decide against further investigation.

On unlabeled trees the CRF is a lot faster.[1] Using the CRF for unlabeled parsing is straightforward. We use two labels: the dummy label and a label X that all constituents start with. We experiment with the CRF for unlabeled trees and observe promising stability that we did not observe with the RNNG, even without any pretraining.

We run into the problem of the derivational ambiguity, however, which halts this investigation. We have noted the problem in chapter 4, but here the problem is even more pronounced because there is just one label besides the dummy label. In this case the problem boils down to the computation of the entropy: with the current setup of the parser forest, we compute the entropy over *derivations*, not the entropy over *trees*. Our optimization exploits this error in a brilliant way: the CRF posterior collapses to a single trivial tree—with all the leaves directly under a single root node—while it learns a perfectly uniform distribution over the *many* derrivations that collapse to that trivial tree. This way the entropy over derivations is high, while we sample the same (collapsed) tree.

The way forward from here is clear—we have already discussed the core solution in chapter 4. Altering the inference algorithms will remove the derivational ambiguity and allows us to compute the correct entropy term. Furthermore, we can learn from the optimization details of Kim et al. [2019], who found that their approach required considerable finetuning to avoid the posterior to collapse to trivial trees.[2] This required separate optimizers for the generative and inference models, annealing the posterior entropy, and freezing the posterior after two epochs [Kim et al., 2019]. This could turn out to be necessary for our approach as well.

6.4. Related work

Our approach is informed by work in neural variational inference with discrete latent variables—in general [Paisley et al., 2012, Mnih and Gregor, 2014, Ranganath et al., 2014, Mnih and Rezende, 2016] and in natural language processing in particular [Miao and Blunsom, 2016, Yin et al., 2018]. The concurrent work Kim et al. [2019] shares many ideas with our work: the use of a CRF inference model, the parametrization of the CRF following Stern et al. [2017a], the exact entropy computation, and even syntactic evaluation on the dataset of Marvin and Linzen [2018]. We consider these similarities to be a remarkable coincidence.

[1] The unlabeled CRF takes around 25 minutes per epoch on the PTB takes around, vesus 4 hours for the labeled CRF with the full 100+ labelset.

[2] Binary right branching trees in their case.

7. Conclusion

We considered neural network language models that incorporate syntactic structure. Central to our discussion was the RNNG, in which the syntactic structure is modelled explicitly in a joint distribution $p(x, y)$ and a language model $p(x)$ is obtained through approximate marginalization over y using a discriminative proposal model. The approximate marginalization is central in the application of the RNNG as language model, and we have introduced a neural CRF parser to investigate the impact of an alternative proposal distribution. As a side gain we obtained a competitive probabilistic parser, and with that a globally normalized probability distribution over trees that has aplications beyond what we discussed. We briefly discussed neural language models that only model x but receive syntactic supervision during training in the form of multitask learning. To gain insight in their comparative syntactic abilities we performed targeted syntactic evaluation using the Syneval dataset, which gave us a detailed breakdown. Finally, we showed how the RNNG and CRF can be combined in semisupervised and unsupervised learning, but must leave the full exploration of those ideas for future work.

We now list the main constributions of this thesis and follow this with suggestions for future work that depart from them.

7.1. Main contributions

The main contributions of this thesis are: (1) the neural CRF parser introduced in chapter 4; (2) the syntactic evaluation of all our models and (3) the comparison of the RNNG to syntactic multitask models in chapter 5; and (4) the semisupervised and unsupervised learning of the RNNG guided by a CRF proposal model in chapter 6.

Neural CRF parser We presented a span factored neural CRF parser by borrowing the span factored approach and neural scoring function from Stern et al. [2017a] and deriving custom inference algorithms from general inside and outside recursions. We showed that the model is a competitive parser that outperforms a discriminative RNNG of the same size by a large margin and moreover appears to deal with the absence of tag information much better. We used the CRF as a proposal distribution for the RNNG, but noted that there is a subtle mismatch between the space of trees modelled by the CRF and the RNNG. This is caused by derivational ambiguity which in turn is caused by our choice of binarization. We showed how the parse forest of the CRF can be altered to deal with this, but must leave the implementation and evaluation to future work. Finally, the prohibitively slow training is a drawback of the model, but we noted that pruning

the labelset—removing mostly labels corresponding to rarely occuring unary chains—results in linear speedup as a result of the $O(n^3|\Lambda|)$ time complexity. The pruning will inevitably affect the accuracy of the model, and future research can investigate this.

Syntactic evaluation of RNNG We performed targeted syntactic evaluation of the RNNG on the Syneval dataset. This gave us a detailed breakdown per model. However, the finegrained evaluation also posed an interpretative challenge, and we additionally opted for a broader view provided by the averaging the results.

Comparison with multitask learning We proposed a novel multitask language model with a side objective inspired by the span scoring function of the CRF, and described a previous model based on CCG supertagging. This method provides an alternative way to bias language models towards syntax, and in at least one category of the Syneval dataset this method outperformed the RNNG (on average).

Semisupervised learning of RNNG We formulated semisupervised and unsupervised estimation of the RNNG using variational inference. We have showed that the CRF posterior allows us to derrive a particularly satisfying version of the ELBO in which the entropy term can be computed exactly, and have argued that the global normalization makes for a distribution that is more amenable to sampling based gradient estimation. Our first attempt at semisupervised learning from pretrained turned out unsuccesful: the CRF is too slow, and with the RNNG we could not prevent the posterior from degenarating to pathological trees. The unlabeled setting proved more promising, but the derivational ambiguity in the CRF prevented us from optimizing the proper objective.

7.2. Future work

We see two directions for future work, both on unsupervised learning of the RNNG with the CRF as posterior. The first continues with the unsupervised learning where this thesis left off, following through with our proposed refinements of the CRF. The second direction that we propose takes the RNNG and CRF into the direction of sparse structured inference.

Unsupervised RNNG with CRF posterior The line of work that departs from where this thesis left off is the further investigation of the CRF proposal for unsupervised learning of the RNNG. That this direction is fruitful has recently been shown by Kim et al. [2019]. But our work differs from theirs because we do not restrict to binary trees and use the original formulation of the RNNG. We have made the first steps in this direction, but were halted by a fundamental mismatch between the support of the RNNG and the CRF. We have described the steps that need to be taken to overcome

this mismatch. We can learn from the insights of Kim et al. [2019] and follow the training strategy that found was required to, including separate optimizers, annealing the posterior, and early stopping.

SparseMAP inference Another direction of interest is to perform unsupervised learning of the generative RNNG with CRF posterior using the recently proposed SparseMAP inference [Niculae et al., 2018a]. SparseMAP allows sparse structured inference with neural networks by inducing sparse posterior distributions over the latent strucutre. This allows fully differentiable training of neural networks with discrete latent structure, and has been used effectively for unsupervised induction of dependency trees [Niculae et al., 2018b]. The key requirement for sparseMAP is that the posterior model permits efficient and exact MAP inference: this is precicely what is provided for our CRF parser by the Viterbi algorithm. And in contrast with the approaches that we have discussed in chapter 6 SparseMAP requires no approximation by sampling. A first sketch of our approach would be to optimize a kind of autoencoding objective

$$\sum_{y \in \mathcal{Y}(x)} p_\theta(x \mid y) q_\lambda(y \mid x),$$

with trees as discrete latent structure. Here q would be the CRF model, and p an adaptation of the joint RNNG in which only the words are modelled, and the structure y is conditioned on.[1]. SparseMAP makes the sum over $\mathcal{Y}(x)$ tractable by inducing sparsity in q, such that most of the terms in the sum are zero. Only for the small number of trees for which the posterior is nonzero do we need to compute $p_\theta(x \mid y)$. This training objective is not probabilistic, and the two conditional models p and q together do not define a language model, but it can be used to for unsupervised tree induction, and as such provide interesting comparison with the approaches based on variational inference.

[1] A straightforward adaptation of the transition system achieves this: all other actions are provided, and only the word in the GEN action needs to be predicted.

A. Implementation

This appendix reports all details about the implementation of the models, including the datasets and their pre-processing, optimization, and hyperparameters. We have made some deliberate choices which we will motivate. As a general rule, we choose simplicity over maximal performance: we use a minimal scheme for dealing with unknown words, we use only word embeddings—which are learned from scratch—and use regular SGD optimization. The reason: to compare the models in a minimal setting so as to maximally focus on the essential modelling differences between the models. All code is available at `github.com/daandouwe/thesis`.

A.1. Data

A.1.1. Datasets

We use three datasets: the Penn Treebank for the parsing models; CCG supertags for the RNNLM-CCG multitask model; and (part of) the One Billion Word Benchmark for unlabeled data in the semisupervised training.

Penn Treebank We use a publicly available version of the Penn Treebank (PTB) that was preprocessed and published by Cross and Huang [2016], and that has since been used in the experiments of Stern et al. [2017a], Kitaev and Klein [2018]. The data is divided along the standard splits of sections 2-21 for training, section 22 for development, and section 23 for testing. The dataset comes with predicted tags, which is a requirement for neural parsers to avoid overfitting, but note however that none of our models use tag information. The dataset is availlable at `https://github.com/jhcross/span-parser/data`.[1]

CCG supertags For the multitask language model with CCG supertagging we use the CCGBank [Hockenmaier and Steedman, 2007] processed by Enguehard et al. [2017] into a word-tag format. This is the dataset also used by Marvin and Linzen [2018]. It follows the same splits of the Penn Treebank as described above, and restricts the size of the tagset from the original 1363 different supertags to 452 supertags that occurred at least ten times, replacing the rest of the tags with a dummy token. The dataset is publicly availlable at `https://github.com/BeckyMarvin/LM_syneval/tree/master/data/ccg_data`.

[1] Although I do not know how it is possible to make the PTB public, given the licensing restrictions of the LDC, I am very thankful that it was done. Now, all the data used in this thesis is publicly availlable.

One Billion Word Benchmark In the experiments on semisupervised learning we use the One Billion Word Benchmark (OBW) dataset [Chelba et al., 2013] to obtain unlabeled data. This is a common dataset for large-scale language modelling [Jozefowicz et al., 2016], and has the advantage that it has sentences seperated by a newline; a requirement for the RNNG language model, which should only be applied to entire sentences and not to longer or shorter segments.[2] The dataset in its entirety is far too large for our purposes, so instead we select sentences from the first section of the training data[3] by selecting the first 100,000 sentences that have at most 40 words. Because the OBW uses slightly different tokenization and standardization of characters than the Penn Treebank we perform a number of processing steps to smooth out these difference. First, we escape all brackets following the Penn Treebank convention (replacing `(` with `-LRB-`) and do the same for the quotation marks (replacing `"` with ``` `` ```). Finally, tokenization of negation is handled differently in the OBW, and we change this to the PTB convention (replacing `don 't` with `do n't`). These simple changes together avoid a lot of incoherences when combining this dataset with the PTB. The dataset is publicly availlable at `http://www.statmt.org/lm-benchmark/`, and scripts for preprocessing are availlable at `github.com/daandouwe/thesis/scripts`.

A.1.2. Vocabularies

We use two types of vocabularies corresponding to the two types of models that we study this thesis: a vocabulary for the discriminative models and a vocabulary for the generative models. The vocabulary used in the discrminative models contains all words in the training data, whereas the vocabulary used in the generative models only includes words that occur at least 2 times in the training data[4]. Finally, in the semisupervised models we construct the vocabulary from the labeled and unlabeled datasets combined. To keep the vocabulary size manageable in this larger dataset we restrict the vocabulary of the generative model to words that occur at leest 3 times.

Unknown words We use a single token for unknown words, and during training replace each word w by this token with probability

$$\frac{1}{1 + \text{freq}(w)},$$

where $\text{freq}(w)$ is the frequency of w in the training data. In this we follow Stern et al. [2017a]. We deviate from Dyer et al. [2016], who use a set of almost 50 tokens each with detailed lexical information about the unknown word in question.[5] This elaborate

[2] For this reason we cannot use the otherwise appealing Wikitext dataset [Merity et al., 2016]: this dataset has sentences grouped into paragraphs.

[3] Section `news.en-00001-of-00100`.

[4] This makes training of the generative model faster, because the softmax normalization involves less terms in the sum, and additionaly avoids the statistical difficulty related to predicting words that occur just once. The discrminative model has neither of these problems, since the words are just conditioned on.

[5] An approach taken from the Berkeley parser [Petrov et al., 2006].

approach is common in parsing but certainly not in language modelling [Dyer et al., 2016], for which reason we opt for the simpler scheme of a single token.

Embeddings All word embeddings are learned from scratch: the embeddings are considered part of the model's parameters and are optimized jointly with the rest of them, starting from random initialization [Glorot and Bengio, 2010]. We surmise that more elaborate embeddings should improve performance of the models, but such investigation is in principle othogonal to our work.[6] We furthermore do not experiment with any kind of subword information in the embeddings and, as noted before, make no use of tags in any of the models. The influence of such embeddings on the discriminate parser of Stern et al. [2017a] is analysed extensively by Gaddy et al. [2018], who investigate all combinations of word, tag, and character-LSTM embeddings, and find that the best model[7] improves on the worst model[8] by only 0.8 F1, which is a relative improvement of just 1%. We believe this justifies our basic approach.

Model	Parameters
Discriminative RNNG	798,078
Generative RNNG	9,610,736
CRF	762,752
CRF (big)	2,337,282
RNNLM	9,308,979
RNNLM-span	9,420,172
RNNLM-CCG	9,498,986

Table A.1.: Number of parameters of all models used.

A.2. Implementation

All our models are implemented in python using the Dynet neural network library [Neubig et al., 2017a], and use automatic batching [Neubig et al., 2017b]. Autobatching enables efficient training of our models, for which manual batching is too difficult.

Optimization All our models are optimized with stochastic gradient-based methods, in which we use mini-batches to compute stochastic approximations of the model's gradient on the entire dataset, and let Dynet compute the gradients using automatic differentiation [Neubig et al., 2017a, Baydin et al., 2018]. For all our supervised models we use regular stochastic gradient descent (SGD), with an initial learning rate of 0.1,

[6] See for example the impact of ELMo embeddings [Peters et al., 2018] on the performance of the parser in Kitaev and Klein [2018] (an adaptation of the chart parser in Stern et al. [2017a]).

[7] A tie between the model that uses all embeddings concatenated and the model that uses just the character-LSTM, both with an F1 of 92.24.

[8] Using only word embeddings, at an F1 of 91.44.

and anneal this by a factor of 2 when the performance on the development set fails to improve. This follows the recommendations of Wilson et al. [2017], who show that on models and datasets similar to ours this method finds good solutions and is more robust against overfitting than methods with adaptive learning rates. This also follows Dyer et al. [2016], who obtain their best RNNG models using this optimizer and learning rate. Our models use dropout on all layers, including recurrent layers, and use weight decay of 10^{-6}. For our semisupervised model we do rely on adaptive gradient methods, and use Adam [Kingma and Ba, 2014] with the default learning rate[9] of 0.001. Adaptive learning are considered more suitable for dealing with the dramatic variability in magnitude of the surrogate obective [Ranganath et al., 2014, Fried and Klein, 2018]. For all supervised models we use minibatches of size 10.

[9] To be precise, this is the value α in Kingma and Ba [2014].

B. Semiring parsing

The derivation of the inference algorithms in chapter 4 makes use of the notion of a *weighted hypergraph* as a compact representation of all parses and their scores [Gallo et al., 1993, Klein and Manning, 2004], and also makes use some of the ideas and notation of *semiring parsing* [Goodman, 1999, Li and Eisner, 2009].

B.1. Hypergraph

We use a *backward hypegraph* to compactly represent all possible trees over a sentence under a grammar, an idea introduced in Klein and Manning [2004]. We call such hypergraph a parse forest.

Definition B.1.1. A *directed hypergraph* is a pair $G = (V, E)$, consisting of a set of nodes V, and a set of directed hyperedges $E \subseteq 2^V \times 2^V$ that connect a set of nodes at the *tail* of the arrow to a set of node at the *head* of the arrow.

Definition B.1.2. A *backward-hypergraph* is a directed hypergraph where the hyperedges $E \subseteq 2^V \times V$ connect to a single node. For a node $v \in V$ the set $I(v) \subseteq E$ denotes the set of edges incoming at v and the set $O(v) \subseteq E$ denotes the set of edges outgoing from v, *i.e.* the edges for which v is in the tail. For an edge e we write $T(e) \subseteq V$ for the set of nodes in the tail of e, and define $H(e) \in V$ as the node at the head of the edge.

Definition B.1.3. A *weighted hypergraph* is any hypergraph G equiped with a weight function $\omega : E \to K$ that to each edge assigns a weight from a weight-set K. Typically K the set of real numbers, or the set of integers.

Definition B.1.4. A *hyperpath* in a backward-hypergraph is a set of edges that connects a single node $v \in V$ at the sink to a set of nodes $W \subseteq V$ upstream. Formally defining a hyperpath is a little cumbersome, but informally, a hyperpath is a set of edges obtained by the following recursive procedure: starting at v, follow a single incoming hyperarc e backwards, save the edge and repeat this for all nodes in $T(e)$; stop when all nodes in W have been encountered. In the context of parsing we also call a hyperpath from the root node to the set of words $\{x_1, \ldots, x_n\}$ a *derivation*: this hyperpath represents a single tree for the sentence.

Example B.1.5. (Parse forest) Figure B.1 shows a fragment of a hypergraph that represents the parse forest over an example sentence. Shown are the two hyperpaths corresponding to the two partially overlapping derivations

(S (NP *The* (ADJP *very hungry*) *cat*) (VP *meows*) .) (B.1)

(S (NP *The* (NP (ADJP *very hungry*) *cat*)) (VP *meows*) .) (B.2)

after collapsing the empty nodes $\varnothing$. The edges that direct to the special node $(S^\dagger, 0, n)$ make the hypergraph *rooted*: each *hyperpath* with the set of all words at the source ends at this one node.

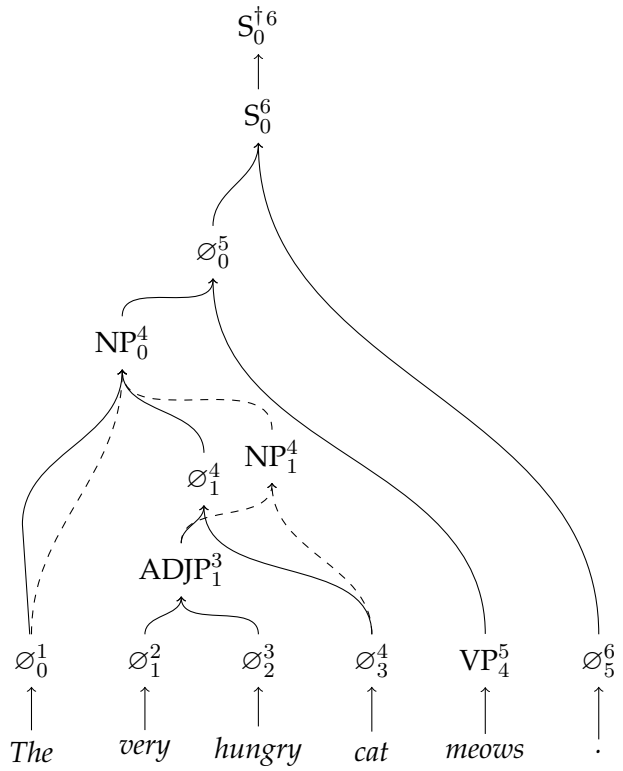


Figure B.1.: A fraction of a parse hypergraph showing two possible parses.

B.2. Semiring

We use *semirings* to compute various quantities of interest over a weighted hypergraph.

Definition B.2.1. A *semiring* is an algebraic structure

$$\mathcal{K} = (\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1}),$$

over a field $\mathbb{K}$, with additive and multiplicative operations $\oplus$ and $\otimes$, and additive and multiplicative identities $\bar{0}$ and $\bar{1}$. A semiring is equivalent to a ring[1], without the requirement of an additive inverse for each element.

Some of the semirings relevant to our discussion are the following:

[1]Perhaps the most common algebraic structure around: the real numbers with regular addition and multiplication form a ring.

Example B.2.2. The *real semiring*

$$(\mathbb{R}_{\geq 0}, +, \times, 0, 1),$$

defined over the nonnegative reals, with regular addition and multiplication is a semiring. Note that the additive inverse is missing for all elements greater than zero (*i.e.* missing the negative reals).

Example B.2.3. The *boolean semiring*

$$(\{\top, \bot\}, \vee, \wedge, \top, \bot),$$

is defined over truth values $\top$ (True, or 1), and $\bot$ (False, or 0). The binary operations are the logical 'and' and 'or' operations.

Example B.2.4. The *log-real semiring*,

$$(\mathbb{R} \cup \{-\infty\}, \oplus, +, -\infty, 0),$$

defined over the reals extended including $-\infty$, with addition defined as the logarithmic sum[2]

$$a \oplus b = \log(e^a + e^b),$$

and multiplication is defined as regular addition.

Example B.2.5. The *max-tropical semiring*, or *Viterbi* semiring[3]

$$(\mathbb{R}_{\geq 0} \cup \{-\infty\}, \max, +, -\infty, 0),$$

is the log-real semiring that uses the $\max$ operator for addition.

B.3. Semiring parsing

A semiring $\mathcal{K}$ can be connected to a weighted hypergraph by defining the function ω over its field $\mathbb{K}$, and by accumulating the weights with its binary operations. When the hypergraph represents a parse forest, we are in the realm of *semiring parsing* [Goodman, 1999].

The key result derived by Goodman [1999] is that many quantities of interest can be computed by a single recursion but over different semirings. First, we establish the relation between the weight function and the structures encoded in the hypergraph by defining the weight of a derivation and the weight of an entire hypergraph.

[2] Also known as *log-sum-exp*, or *log-add-exp*.
[3] This naming will become clear.

Definition B.3.1. (Hypergraph weights) Let $G_\omega = (V, E, \omega)$ be a weighted hypergraph, with ω defined over a semiring $\mathcal{K}$. We define the weight of the derrivation $D \subseteq E$ as the product of the weights of the edges:

$$\bigotimes_{e \in D} \omega(e). \tag{B.3}$$

Let $\mathcal{D} \subseteq 2^E$ be the set of all derivations in the hypergraph G. Then the total weight of the hypergraph under ω is defined as the sum of the weights of all the derrivations in it:

$$\bigoplus_{D \in \mathcal{D}} \bigotimes_{e \in D} \omega(e). \tag{B.4}$$

Example B.3.2. (CRF parser) The CRF parser assigns a score $\Psi(x, y) \geq 0$ by factorizing over parts of $y = \{y_a\}_{a=1}^{A}$ as a product of potentials $\psi(x, y_a) \geq 0$ of the parts. The parts y_a are the edges in the hyperpath that create the derivation y. In our CRF, thus, the weight function ω is given by the function ψ; the function Ψ is equivalent to equation B.3; and the sum over $\mathcal{Y}(x)$ is equivalent to equation B.4.

B.3.1. Inside and outside recursions

We follow the exposition of Li and Eisner [2009], but the results where first derived in Goodman [1999].

Definition B.3.3. The *inside value* $\alpha(v)$ at a node $v \in V$ accumulates the weight of all the paths that converge at that node. The accumulation is relative to a semiring, and is defined as

$$\alpha(v) = \begin{cases} \bar{1} & \text{if } I(v) = \varnothing, \\ \bigoplus_{e \in I(v)} \omega(e) \otimes \bigotimes_{u \in T(e)} \alpha(u) & \text{otherwise.} \end{cases}$$

The value $\alpha(v)$ at the root node v is the sum of the weight of all the derivations in the hypergraph. The recursion can solved by visiting the nodes in V in topological order.

Definition B.3.4. The *outside value* $\beta(v)$ at a node $v \in V$ accumulates over the weights of all the paths that head out from v. The accumulation is relative to a semiring, and is defined as:

$$\beta(v) = \begin{cases} \bar{1} & \text{if } O(v) = \varnothing, \\ \bigoplus_{e \in O(v)} \omega(w) \otimes \beta(H(e)) \otimes \bigotimes_{\substack{w \in T(e) \\ w \neq u}} \alpha(w) & \text{otherwise.} \end{cases}$$

The recursion can solved by visiting the nodes in V in reverse topological order.

B.3.2. Instantiated recursions

By instanstiating the inside and outside recursions with different semirings we solve a whole range of problems.

Example B.3.5. (Inside and outside values) When we intantiate the inside recursion and the outside recursion with the *real semiring*, the algorithms reduce to the classical inside-outside algorithm [Baker, 1979]. The values $\alpha(v)$ and $\beta(v)$ are the inside and outside values. In particular, the value of α at the root is the normalizer:

$$\alpha(\mathsf{S}^\dagger, 0, n) = Z(x).$$

In our CRF parser, the function ω is given by the nonnegative function Ψ.

Example B.3.6. (Logartihmic domain) When we are concerned with numerical stability, or we only need the logarithm of the quantities of interest, we can use the *log-real semiring*. The values $\alpha(v)$ and $\beta(v)$ now give the log-inside and log-outside values respectively. In particular, the value α at the root now gives the lognormalizer:

$$\alpha(\mathsf{S}^\dagger, 0, n) = \log Z(x).$$

The semiring addition $a \oplus b = \log(e^a + e^b)$ is made numerically stable by writing

$$\log(e^a + e^b) = a + \log(1 + e^{b-a}) = b + \log(1 + e^{a-b})$$

and choosing the expression with the smaller exponent. In the CRF, the function ω is given by the composed function $\log \circ \Psi$.

Example B.3.7. (Viterbi weight) When we intantiate the inside recursion with the *max-tropical semiring* we get the recursion that computes at each node the subtree of maximum weight. The value α at the root is the weight of the derrivation with the maximum weight

$$\alpha(\mathsf{S}^\dagger, 0, n) = \log \Psi(x, \hat{y}),$$

where $\hat{y}$ is the Viterbi tree. Normalizing the score with the lognormalizer gives the log-probability

$$\log p(\hat{y} \mid x) = \alpha(\mathsf{S}^\dagger, 0, n) - \log Z(x).$$

Hence the alternative name *Viterbi semiring*.

Example B.3.8. (Viterbi derivation) The Viterbi semiring derives the *weight* of the best tree. Replacing the max in the Viterbi semiring with an argmax derives the best tree itself:

$$\alpha(\mathsf{S}^\dagger, 0, n) = \hat{y}, \qquad \hat{y} \triangleq \operatorname*{arg\,max}_{y \in \mathcal{Y}(x)} p(y \mid x).$$

Roughly spearking, because although this idea can be made precise by a constructing the *Viterbi-derivation semiring* [Goodman, 1999] over the set of possible derrivations, with corresponding binary operations and set-typed identity elements $\bar{0}$ and $\bar{1}$, it is a litle more complicated than that[4].

Example B.3.9. (Recognition) When we intantiate the inside recursion with the *boolean semiring* we get the recursion that recognizes wheter a hyperpath exists from the words spanned by the node:

$$\alpha(\mathrm{S}^{\dagger}, 0, n) = \begin{cases} \top & \text{if } x \in L(G), \\ \bot & \text{otherwise.} \end{cases}$$

The value at the root tells wether the sentence has at least one parse. Due to the trivial grammar used this will always be $\top$ in our CRF.

[4] And most of all, a bit cumbersome.

C. Variational inference

This appendix contain derivations and background used in chapter 6 on semisupervised learning. We derive the score function estimator, describe variance reduction of the estimator by using control variates and baselines, and detail how the estimator is implemented in an automatic differentiation toolkit.

C.1. Score function estimator

In this section we derive the score function estimator

$$\nabla_\lambda \mathbb{E}_q[L(x,y)] = \mathbb{E}_q[L(x,y)\nabla_\lambda \log q_\lambda(y|x)], \tag{C.1}$$

where $L(x,y)$ is some function of x and y. We derive it for the case where

$$L(x,y) \triangleq \log p_\theta(x,y) - \log q_\lambda(y|x),$$

which is the form used for the RNNG posterior, and note that the simpler form for the CRF posterior follows from it. The line by line derivation is given by

$$\begin{aligned}
\nabla_\lambda \mathbb{E}_q[L(x,y)] &= \nabla_\lambda \mathbb{E}_q[\log p_\theta(x,y) - \log q_\lambda(y|x)] \\
&= \nabla_\lambda \sum_{y\in\mathcal{Y}(x)} q_\lambda(y|x)\log p_\theta(x,y) - q_\lambda(y|x)\log q_\lambda(y|x) \\
&= \sum_{y\in\mathcal{Y}(x)} \nabla_\lambda q_\lambda(y|x)\log p_\theta(x,y) - \nabla_\lambda q_\lambda(y|x)\log q_\lambda(y|x) - q_\lambda(y|x)\nabla_\lambda \log q_\lambda(y|x) \\
&= \sum_{y\in\mathcal{Y}(x)} \nabla_\lambda q_\lambda(y|x)\log p_\theta(x,y) - \nabla_\lambda q_\lambda(y|x)\log q_\lambda(y|x) \\
&= \sum_{y\in\mathcal{Y}(x)} L(x,y)\nabla_\lambda q_\lambda(y|x) \\
&= \sum_{y\in\mathcal{Y}(x)} L(x,y) q_\lambda(y|x)\nabla_\lambda \log q_\lambda(y|x) \\
&= \mathbb{E}_q[L(x,y)\nabla_\lambda \log q_\lambda(y|x)].
\end{aligned}$$

In this derivation we used the identity

$$\nabla_\lambda q_\lambda(y|x) = q_\lambda(y|x)\nabla_\lambda \log q_\lambda(y|x),$$

which follows from the derivative

$$\nabla_\lambda \log q_\lambda(y|x) = \nabla_\lambda q_\lambda(y|x) q_\lambda(y|x)^{-1},$$

and finally the fact that

$$\begin{aligned}
\sum_{y \in \mathcal{Y}(x)} q_\lambda(y|x) \nabla_\lambda \log q_\lambda(y|x) &= \sum_{y \in \mathcal{Y}(x)} q_\lambda(y|x) \frac{\nabla_\lambda q_\lambda(y|x)}{q_\lambda(y|x)} \\
&= \sum_{y \in \mathcal{Y}(x)} \nabla_\lambda q_\lambda(y|x) \\
&= \nabla_\lambda \sum_{y \in \mathcal{Y}(x)} q_\lambda(y|x) \\
&= \nabla_\lambda 1 \\
&= 0.
\end{aligned}$$

C.2. Variance reduction

The score function estimator is unbiased, but is known to have high variance, often too much to be useful in practice [Paisley et al., 2012]. Two effective methods to counter this are control variates and baselines [Ross, 2006], and in this section we describe what they are and why they work. For the rest of this section we consider ourselves with expectations of the general form

$$\mu \triangleq \mathbb{E}[f(X)]$$

that we estimate as

$$\hat{\mu} = \frac{1}{K} \sum_{k=1}^{K} f(X^{(k)})$$

using samples $X^{(1)}, \ldots, X^{(K)}$.

C.2.1. Control variates

A control variate is another function of X that we subtract from $f(X)$, redefining the estimator. When g and f are correlated the resulting estimator has lower variance. For this, we consider a function g with known expectation

$$\mu_g = \mathbb{E}[g(X)]$$

and define a new function $\hat{f}$ as

$$\hat{f}(X) \triangleq f(X) - g(X) + \mu_g.$$

We note that this function is also an estimator for μ because

$$\begin{aligned}\mathbb{E}[\hat{f}(X)] &= \mathbb{E}[f(X)] - \mu_g + \mu_g \\ &= \mathbb{E}[f(X)],\end{aligned}$$

and we can compute the variance of the new function as

$$\begin{aligned}\operatorname{Var}[\hat{f}(X)] &= \mathbb{E}[(f(X) - g(X) + \mu_g) - \mu)^2] \\ &= \mathbb{E}[(f(X) - g(X) + \mu_g)^2] - 2\,\mathbb{E}[(f(X) - g(X) + \mu_g)\mu] + \mathbb{E}[\mu^2] \\ &= \mathbb{E}[(f(X) - g(X) + \mu_g)^2] - 2\,\mathbb{E}[(f(X) - g(X) + \mu_g)]\mu + \mu^2 \\ &= \mathbb{E}[(f(X) - g(X) + \mu_g)^2] - 2\mu^2 + \mu^2 \\ &= \mathbb{E}[f(X)^2 + g(X)^2 + \mu_g^2 - 2f(X)g(X) + 2f(X)\mu_g - 2g(X)\mu_g] - \mu^2 \\ &= \mathbb{E}[f(X)^2] - \mathbb{E}[f(X)]^2 \\ &\quad - 2(\mathbb{E}[f(X)g(X)] - \mathbb{E}[f(X)]\,\mathbb{E}[g(X)]) \\ &\quad + \mathbb{E}[g(X)^2] - \mathbb{E}[g(X)]^2 \\ &= \operatorname{Var}[f(X)] - 2\operatorname{Cov}[f(X), g(X)] + \operatorname{Var}[g(X)].\end{aligned}$$

This means we can get a reduction in variance whenever

$$\operatorname{Cov}[f(X), g(X)] > \frac{1}{2}\operatorname{Var}[g(X)], \tag{C.2}$$

which will be the case when $f(X)$ and $g(X)$ are correlated.

The function g is called a control variate, and can be chosen in any way as long as the expectation can be computed exactly. A more general formulation of the control variate is to introduce a scalar a

$$\hat{f}(X) \triangleq f(X) - a(g(X) - \mathbb{E}[g(X)]),$$

which can be used to maximize the correlation between f and g. This function $\hat{f}$ has variance

$$\operatorname{Var}[\hat{f}(X)] = \operatorname{Var}[f(X)] - 2a\operatorname{Cov}[f(X), g(X)] + a^2\operatorname{Var}[g(X)].$$

We take a derivative of this with respect to a

$$\frac{d}{da}\operatorname{Var}[\hat{f}(X)] = -2\operatorname{Cov}[f(X), g(X)] + 2a\operatorname{Var}[g(X)],$$

set to zero, solve for a to obtain

$$a = \frac{\operatorname{Cov}[f(X), g(X)]}{\operatorname{Var}[g(X)]} \tag{C.3}$$

as optimal choice for a.

Plugging in this solution into the expression for $\mathrm{Var}[\hat{f}(X)]$ and dividing by $\mathrm{Var}[f(X)]$ we get

$$\frac{\mathrm{Var}[\hat{f}(X)]}{\mathrm{Var}[f(X)]} = 1 - \frac{\mathrm{Cov}[f(X), g(X)]}{\mathrm{Var}[f(X)]\,\mathrm{Var}[g(X)]} \tag{C.4}$$

$$= 1 - \mathrm{corr}^2[f(X), g(X)]. \tag{C.5}$$

This shows that given the optimal choice of a the reduction in variance is directly determined by the correlation between $f(X)$ and $g(X)$.

Combined this gives the new estimator

$$\mathbb{E}[f(X)] = \mathbb{E}[\hat{f}(X)] \approx \frac{1}{K}\sum_{k=1}^{K}[f(X^{(k)}) - ag(X^{(k)})] + \mu_g,$$

Example C.2.1. [Ross, 2006] Suppose we want to use simulation to determine $\mathbb{E}[e^X]$ with $X \sim \text{Uniform}(0, 1)$. Of course, we can compute analytically that

$$\mathbb{E}[e^X] = \int_0^1 e^x dx = e - 1$$

A natural control variate to use in this case is the random variable X itself, $g(X) \triangleq X$, which gives the new estimator

$$\begin{aligned}\hat{f}(X) &= f(X) - g(X) + \mathbb{E}[g(X)] \\ &= e^X - X + \frac{1}{2}.\end{aligned}$$

Then to compute the reduction in variance with this new estimator, we first note that

$$\begin{aligned}\mathrm{Cov}(e^X, X) &= \mathbb{E}[Xe^X] - \mathbb{E}[X]\,\mathbb{E}[e^X] \\ &= \int_0^1 xe^x dx - \frac{e-1}{2} \\ &= 1 - \frac{e-1}{2} \approx 0.14086,\end{aligned}$$

that

$$\begin{aligned}\mathrm{Var}[e^X] &= \mathbb{E}[e^{2X}] - (\mathbb{E}[e^X])^2 \\ &= \int_0^1 e^{2x} dx - (1 - e^x)^2 \\ &= \frac{e^2 - 1}{2} - (1 - e^x)^2 \approx 0.2420,\end{aligned}$$

and that

$$\begin{aligned}\mathrm{Var}[X] &= \mathbb{E}[X^2] - (\mathbb{E}[X])^2 \\ &= \int_0^1 x^2 dx - \frac{1}{4} \\ &= \frac{1}{3} - \frac{1}{4} = \frac{1}{12}.\end{aligned}$$

When we choose a as in formula C.3 we can use formula C.4 to compute that

$$\begin{aligned}\frac{\mathrm{Var}[\hat{f}(X)]}{\mathrm{Var}[f(X)]} &= 1 - \frac{(0.14086)^2}{\frac{0.2420}{12}} \\ &\approx 0.0161.\end{aligned}$$

This is a reduction of over 98%!

C.2.2. Baseline

Using a control variate requires us to know $\mathbb{E}[g(X)]$, which is unlikely for the distributions that we consider in this work. An alternative that does not have this requirement is a *baseline*, which is a scalar value b that unlike the function g does not depend on the random variable X. Using the baseline we formulate the estimator

$$\mathbb{E}[f(X) - b] + b \tag{C.6}$$

$$\approx b + \frac{1}{K}\sum_{k=1}^{K} f(X^{(k)}) - b, \tag{C.7}$$

which is trivially also an estimator for $f(X)$, because adding and subtracting b does not change anything. Why would this do anything? Well, in the case that the distribution of X is parametrised by some θ and we compute gradients with respect to the estimator we see that the

$$\nabla_\theta \mathbb{E}[f(X)] = \nabla_\theta \mathbb{E}[f(X) - b] + b \tag{C.8}$$

$$= \mathbb{E}[(f(X) - b)\nabla_\theta \log p(X)] \tag{C.9}$$

$$=\approx \frac{1}{K}\sum_{k=1}^{K}(f(X^{(k)}) - b)\log p(X^{(k)}), \tag{C.10}$$

and again we get a reduction in variance if b correlates well with $f(X)$.

C.3. Optimization

We use automatic differentiation [Baydin et al., 2018] to obtain all our gradients. In order to obtain the gradients in formula C.1 using this method we rewrite it in the form

of a *surrogate objective* [Schulman et al., 2015]:

$$\mathcal{L}_{\text{SURR}}(\theta, \lambda) = \frac{1}{K} \sum_{k=1}^{K} \log q_\lambda(x|y^{(k)}) \text{BLOCKGRAD}(L(x, y^{(k)})). \tag{C.11}$$

The function BLOCKGRAD detaches a node from its upstream computation graph, removing its dependence on the parameters. To roughly illustrate this, let f be function (computed by a node in the computation graph) with parameters θ and input x, then

$$\text{BLOCKGRAD}(f_\theta(x)) \triangleq f(x),$$

such that

$$\nabla_\theta \text{BLOCKGRAD}(f_\theta(x)) = \nabla_\theta f(x) = 0.$$

Automatic differentiation of equation C.11 with respect to λ will give us the exact expression we are looking for

$$\nabla_\lambda \mathcal{L}_{\text{SURR}}(\theta, \lambda) = \frac{1}{K} \sum_{k=1}^{K} L(x, y^{(k)}) \nabla_\lambda \log q_\lambda(x|y^{(k)}),$$

hence the adjective *surrogate*.

D. Syneval dataset

The following list is the complete set of categories that are evaulated in the Syneval dataset, taken from Marvin and Linzen [2018] with slight alterations. There is a slight ambiguity in the category of negative polarity items, with two different types on constructions, one of sentences starting with *most* and the other of sentences starting with *most*, combined into one positive set; we choose to separate them into two different categories. For the full list of lexical items used to build variants of these constructions we refer the reader to Marvin and Linzen [2018].

1. **Simple agreement**:
    a) The farmer *smiles*.
    b) *The farmer *smile*.
2. **Agreement in a sentential complement**:
    a) The mechanics said the author *laughs*.
    b) *The mechanics said the author *laugh*.
3. **Agreement in short VP coordination**:
    a) The authors laugh and *swim*.
    b) *The authors laugh and *swims*.
4. **Agreement in long VP coordination**:
    a) The author knows many different foreign languages and *enjoys* playing tennis with colleagues.
    b) *The author knows many different foreign languages and *enjoy* playing tennis with colleagues.
5. **Agreement across a prepositional phrase**:
    a) The author next to the guards *smiles*.
    b) *The author next to the guards *smile*.
6. **Agreement across a subject relative clause**:
    a) The author that likes the security guards *laughs*.
    b) *The author that likes the security guards *laugh*.
7. **Agreement across an object relative**:
    a) The movies that the guard likes *are* good.
    b) *The movies that the guard likes *is* good.

8. **Agreement across an object relative (no *that*)**:
    a) The movies the guard likes *are* good.
    b) *The movies the guard likes *is* good.
9. **Agreement in an object relative**:
    a) The movies that the guard *likes* are good.
    b) *The movies that the guard *like* are good.
10. **Agreement in an object relative (no *that*)**:
    a) The movies the guard *likes* are good.
    b) *The movies the guard *like* are good.
11. **Simple reflexive anaphora**:
    a) The author injured *himself*.
    b) *The author injured *themselves*.
12. **Reflexive in sentential complement**:
    a) The mechanics said the author hurt *himself*.
    b) *The mechanics said the author hurt *themselves*.
13. **Reflexive across a relative clause**:
    a) The author that the guards like injured *himself*.
    b) *The author that the guards like injured *themselves*.
14. **Simple NPI**:
    a) *No* authors have ever been famous.
    b) **Most* authors have ever been famous.
15. **Simple NPI (*the*)**:
    a) *No* authors have ever been popular.
    b) **The* authors have ever been popular.
16. **NPI across a relative clause**:
    a) *No* authors that *the* guards like have ever been famous.
    b) **Most* authors that *no* guards like have ever been famous.
17. **NPI across a relative clause (*the*)**:
    a) *No* authors that *the* guards like have ever been famous.
    b) **The* authors that *no* guards like have ever been famous.

Bibliography

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 the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2442–2452. Association for Computational Linguistics, 2016.

James K Baker. Trainable grammars for speech recognition. *The Journal of the Acoustical Society of America*, 65(S1):S132–S132, 1979.

Miguel Ballesteros, Yoav Goldberg, Chris Dyer, and Noah A. Smith. Training with exploration improves a greedy stack lstm parser. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2005–2010. Association for Computational Linguistics, 2016.

Srinivas Bangalore and Aravind K Joshi. Supertagging: An approach to almost parsing. *Computational linguistics*, 25(2):237–265, 1999.

Atilim Gunes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of Marchine Learning Research*, 18:1–43, 2018.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

Ezra Black, Steven Abney, Dan Flickenger, Claudia Gdaniec, Ralph Grishman, Philip Harrison, Donald Hindle, Robert Ingria, Frederick Jelinek, Judith Klavans, et al. A procedure for quantitatively comparing the syntactic coverage of english grammars. In *Speech and Natural Language: Proceedings of a Workshop Held at Pacific Grove, California, February 19-22, 1991*, 1991.

D. M. Blei, A. Kucukelbir, and J. D. McAuliffe. Variational inference: A review for statisticians. *ArXiv e-prints*, January 2016.

Jonathan R Brennan, Edward P Stabler, Sarah E Van Wagenen, Wen-Ming Luh, and John T Hale. Abstract linguistic structure correlates with temporal activity during naturalistic comprehension. *Brain and language*, 157:81–94, 2016.

Jan Buys and Phil Blunsom. A bayesian model for generative transition-based dependency parsing. In *Proceedings of the Third International Conference on Dependency Linguistics, DepLing 2015, August 24-26 2015, Uppsala University, Uppsala, Sweden*, pages 58–67, 2015a.

Jan Buys and Phil Blunsom. Generative incremental dependency parsing with neural networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, volume 2, pages 863–869, 2015b.

Jan Buys and Phil Blunsom. Neural syntactic generative models with exact marginalization. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, volume 1, pages 942–952, 2018.

A. Carnie. *Constituent Structure*. Oxford linguistics. Oxford University Press, 2010. ISBN 9780199583461.

Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

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

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*, 2013.

Ciprian Chelba, Mohammad Norouzi, and Samy Bengio. N-gram language modeling using recurrent neural network estimation. *arXiv preprint arXiv:1703.10724*, 2017.

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

Noam Chomsky. Rules and representations. *Behavioral and brain sciences*, 3(1):1–15, 1980.

Morten H Christiansen, Christopher M Conway, and Luca Onnis. Similar neural correlates for language and sequential learning: evidence from event-related brain potentials. *Language and cognitive processes*, 27(2):231–256, 2012.

Michael Collins. Head-driven statistical models for natural language parsing. *Computational linguistics*, 29(4):589–637, 2003.

Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537, 2011.

Christopher M Conway and David B Pisoni. Neurocognitive basis of implicit learning of sequential structure and its relation to language processing. *Annals of the New York Academy of Sciences*, 1145(1):113–131, 2008.

James Cross and Liang Huang. Span-based constituency parsing with a structure-label system and provably optimal dynamic oracles. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1–11. Association for Computational Linguistics, 2016.

Timothy Dozat and Christopher D Manning. Deep biaffine attention for neural dependency parsing. *arXiv preprint arXiv:1611.01734*, 2016.

Greg Durrett and Dan Klein. Neural crf parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 302–312. Association for Computational Linguistics, 2015.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. Recurrent neural network grammars. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 199–209. Association for Computational Linguistics, 2016.

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. Association for Computational Linguistics.

Ahmad Emami and Frederick Jelinek. A neural syntactic language model. *Machine learning*, 60(1-3):195–227, 2005.

Émile Enguehard, Yoav Goldberg, and Tal Linzen. Exploring the syntactic abilities of rnns with multi-task learning. In *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017)*, pages 3–14. Association for Computational Linguistics, 2017.

Martin BH Everaert, Marinus AC 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, Alex Kleeman, and Christopher D. Manning. Efficient, feature-based, conditional random field parsing. In *Proceedings of ACL-08: HLT*, pages 959–967. Association for Computational Linguistics, 2008.

Stefan L Frank, Rens Bod, and Morten H Christiansen. How hierarchical is language use? *Proceedings of the Royal Society B: Biological Sciences*, 279(1747):4522–4531, 2012.

Daniel Fried and Dan Klein. Policy gradient as a proxy for dynamic oracles in constituency parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 469–476. Association for Computational Linguistics, 2018.

Michael C. Fu. Gradient estimation. In Barry L. Nelson Edited by Shane G. Henderson, editor, *Handbooks in Operations Research and Management Science (Volume 13)*, chapter 19, pages 575–616. Elsevier, 2006.

David Gaddy, Mitchell Stern, and Dan Klein. What's going on in neural constituency parsers? an analysis. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 999–1010. Association for Computational Linguistics, 2018.

Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in neural information processing systems*, pages 1019–1027, 2016.

Giorgio Gallo, Giustino Longo, Stefano Pallottino, and Sang Nguyen. Directed hypergraphs and applications. *Discrete applied mathematics*, 42(2-3):177–201, 1993.

Anastasia Giannakidou. Negative and positive polarity items: Variation, licensing, and compositionality. *The Handbook of Natural Language Meaning (second edition). Mouton de Gruyter, Berlin*, 2011.

Maureen Gillespie and Neal J Pearlmutter. Hierarchy and scope of planning in subject–verb agreement production. *Cognition*, 118(3):377–397, 2011.

Maureen Gillespie and Neal J Pearlmutter. Against structural constraints in subject–verb agreement production. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 39(2):515, 2013.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

Yoav Goldberg and Joakim Nivre. Training deterministic parsers with non-deterministic oracles. *Transactions of the Association for Computational Linguistics*, 1 (Oct):403–414, 2013.

Joshua Goodman. Semiring parsing. *Computational Linguistics*, 25(4):573–605, 1999.

Kristina Gulordava, Piotr Bojanowski, Edouard Grave, Tal Linzen, and Marco Baroni. Colorless green recurrent networks dream hierarchically. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1195–1205. Association for Computational Linguistics, 2018.

John Hale. A probabilistic earley parser as a psycholinguistic model. In *Proceedings of the second meeting of the North American Chapter of the Association for Computational Linguistics on Language technologies*, pages 1–8. Association for Computational Linguistics, 2001.

John Hale, Chris Dyer, Adhiguna Kuncoro, and Jonathan Brennan. Finding syntax in human encephalography with beam search. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2727–2736. Association for Computational Linguistics, 2018.

David Hall, Greg Durrett, and Dan Klein. Less grammar, more features. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 228–237, 2014.

He He, Jason Eisner, and Hal Daume. Imitation learning by coaching. In *Advances in Neural Information Processing Systems*, pages 3149–3157, 2012.

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.

Rodney D. Huddleston and Geoffrey K. Pullum. *The Cambridge Grammar of the English Language*, April 2002.

Richard Hudson. *An introduction to word grammar*. Cambridge University Press, 2010.

Frederick Jelinek. Information extraction from speech and text, chapter 8, 1997.

MichaelI. Jordan, Zoubin Ghahramani, TommiS. Jaakkola, and LawrenceK. Saul. An introduction to variational methods for graphical models. *Machine Learning*, 37(2): 183–233, 1999.

Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*, 2016.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*, 2014.

Yoon Kim, Carl Denton, Luong Hoang, and Alexander M Rush. Structured attention networks. *arXiv preprint arXiv:1702.00887*, 2017.

Yoon Kim, Alexander M Rush, Lei Yu, Adhiguna Kuncoro, Chris Dyer, and Gábor Melis. Unsupervised recurrent neural network grammars. *arXiv preprint arXiv:1904.03746*, 2019.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

Nikita Kitaev and Dan Klein. Constituency parsing with a self-attentive encoder. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2676–2686. Association for Computational Linguistics, 2018.

Dan Klein and Christopher D. Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*, ACL '03, pages 423–430, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.

Dan Klein and Christopher D Manning. Parsing and hypergraphs. In *New developments in parsing technology*, pages 351–372. Springer, 2004.

Reinhard Kneser and Hermann Ney. Improved backing-off for m-gram language modeling. In *icassp*, volume 1, page 181e4, 1995.

Alp Kucukelbir, Dustin Tran, Rajesh Ranganath, Andrew Gelman, and David M Blei. Automatic differentiation variational inference. *The Journal of Machine Learning Research*, 18(1):430–474, 2017.

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 the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 1249–1258. Association for Computational Linguistics, 2017.

Adhiguna Kuncoro, Chris Dyer, John Hale, Dani Yogatama, Stephen Clark, and Phil Blunsom. Lstms can learn syntax-sensitive dependencies well, but modeling structure makes them better. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1426–1436. Association for Computational Linguistics, 2018.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 282–289, 2001.

Roger Levy. Expectation-based syntactic comprehension. *Cognition*, 106(3):1126–1177, 2008.

Zhifei Li and Jason Eisner. First- and second-order expectation semirings with applications to minimum-risk training on translation forests. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 40–51. Association for Computational Linguistics, 2009.

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.

Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. The penn treebank: annotating predicate argument structure. In *Proceedings of the workshop on Human Language Technology*, pages 114–119. Association for Computational Linguistics, 1994.

Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.

Rebecca Marvin and Tal Linzen. Targeted syntactic evaluation of language models. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1192–1202. Association for Computational Linguistics, 2018.

R Thomas McCoy, Tal Linzen, and Robert Frank. Revisiting the poverty of the stimulus: hierarchical generalization without a hierarchical bias in recurrent neural networks. *arXiv preprint arXiv:1802.09091*, 2018.

Gábor Melis, Chris Dyer, and Phil Blunsom. On the state of the art of evaluation in neural language models. *arXiv preprint arXiv:1707.05589*, 2017.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Yishu Miao and Phil Blunsom. Language as a latent variable: Discrete generative models for sentence compression. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 319–328. Association for Computational Linguistics, 2016.

Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

Andriy Mnih and Karol Gregor. Neural variational inference and learning in belief networks. In *ICML*, 2014.

Andriy Mnih and Danilo J. Rezende. Variational inference for monte carlo objectives. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 2188–2196. JMLR.org, 2016.

Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, et al. Dynet: The dynamic neural network toolkit. *arXiv preprint arXiv:1701.03980*, 2017a.

Graham Neubig, Yoav Goldberg, and Chris Dyer. On-the-fly operation batching in dynamic computation graphs. In *Advances in Neural Information Processing Systems*, pages 3971–3981, 2017b.

Vlad Niculae, Andre Martins, Mathieu Blondel, and Claire Cardie. SparseMAP: Differentiable sparse structured inference. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3799–3808, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018a. PMLR.

Vlad Niculae, André F. T. Martins, and Claire Cardie. Towards dynamic computation graphs via sparse latent structure. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 905–911, Brussels, Belgium, October-November 2018b. Association for Computational Linguistics.

Joakim Nivre. Dependency grammar and dependency parsing. *MSI report*, 5133(1959): 1–32, 2005.

John Paisley, David Blei, and Michael Jordan. Variational bayesian inference with stochastic search. In John Langford and Joelle Pineau, editors, *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, ICML '12, pages 1367–1374, New York, NY, USA, July 2012. Omnipress.

Adam Pauls and Dan Klein. Large-scale syntactic language modeling with treelets. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 959–968. Association for Computational Linguistics, 2012.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proc. of NAACL*, 2018.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 433–440. Association for Computational Linguistics, 2006.

Rajesh Ranganath, Sean Gerrish, and David Blei. Black box variational inference. In *Artificial Intelligence and Statistics*, pages 814–822, 2014.

Steven J. Rennie, Etienne Marcheret, Youssef Mroueh, Jarret Ross, and Vaibhava Goel. Self-critical sequence training for image captioning. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1179–1195, 2017.

Brian Roark. Probabilistic top-down parsing and language modeling. *Computational linguistics*, 27(2):249–276, 2001.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, September 1951.

Ronald Rosenfeld. A maximum entropy approach to adaptive statistical language modelling. *Computer Speech Language*, 10(3):187 – 228, 1996. ISSN 0885-2308.

Sheldon M. Ross. *Simulation, Fourth Edition*. Academic Press, Inc., Orlando, FL, USA, 2006.

John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. Gradient estimation using stochastic computation graphs. In *Advances in Neural Information Processing Systems*, pages 3528–3536, 2015.

Satoshi Sekine and Michael Collins. Evalb bracket scoring program. *URL: http://www. cs. nyu. edu/cs/projects/proteus/evalb*, 1997.

Rico Sennrich. How grammatical is character-level neural machine translation? assessing mt quality with contrastive translation pairs. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 376–382. Association for Computational Linguistics, 2017.

Khalil Sima'an. Computational complexity of probabilistic disambiguation. *Grammars*, 5(2):125–151, 2002.

Noah A Smith. Adversarial evaluation for models of natural language. *arXiv preprint arXiv:1207.0245*, 2012.

Anders Søgaard and Yoav Goldberg. Deep multi-task learning with low level tasks supervised at lower layers. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 231–235. Association for Computational Linguistics, 2016.

Mitchell Stern, Jacob Andreas, and Dan Klein. A minimal span-based neural constituency parser. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 818–827, Vancouver, Canada, July 2017a. Association for Computational Linguistics.

Mitchell Stern, Daniel Fried, and Dan Klein. Effective inference for generative neural parsing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, page 1695–1700. Association for Computational Linguistics, 2017b.

Swabha Swayamdipta, Sam Thomson, Kenton Lee, Luke Zettlemoyer, Chris Dyer, and Noah A. Smith. Syntactic scaffolds for semantic structures. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3772–3782. Association for Computational Linguistics, 2018.

Lucien Tesnière. Elements of structural syntax, translated by timothy osborne and sylvain kahane, 1959.

Ivan Titov and James Henderson. A latent variable model for generative dependency parsing. In *Proceedings of the 10th International Conference on Parsing Technologies*, IWPT '07, pages 144–155, Stroudsburg, PA, USA, 2007. Association for Computational Linguistics. ISBN 978-1-932432-90-9.

Ke Tran, Arianna Bisazza, and Christof Monz. The importance of being recurrent for modeling hierarchical structure. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4731–4736. Association for Computational Linguistics, 2018.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

Andreas Vlachos. An investigation of imitation learning algorithms for structured prediction. In *European Workshop on Reinforcement Learning*, pages 143–154, 2013.

Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. Neural network acceptability judgments. *arXiv preprint arXiv:1805.12471*, 2018.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems*, pages 4148–4158, 2017.

Ming Xiang, Brian Dillon, and Colin Phillips. Illusory licensing effects across dependency types: Erp evidence. *Brain and Language*, 108(1):40 – 55, 2009.

Pengcheng Yin, Chunting Zhou, Junxian He, and Graham Neubig. Structvae: Tree-structured latent variable models for semi-supervised semantic parsing. *arXiv preprint arXiv:1806.07832*, 2018.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

Yuan Zhang and David Weiss. Stack-propagation: Improved representation learning for syntax. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1557–1566. Association for Computational Linguistics, 2016.

Geoffrey Zweig and Christopher JC Burges. The microsoft research sentence completion challenge. Technical report, Citeseer, 2011.